**2nd International Workshop on**

**Computer Graphics, Computer Vision and Mathematics**

in co-operation with

**EUROGRAPHICS**

# GraVisMa 2010

**Workshop Proceedings**

***Edited by***

**Vaclav Skala**, University of West Bohemia, Czech Republic
**Eckhard M. Hitzer**, University of Fukui, Japan

## 2nd International Workshop on

## Computer Graphics, Computer Vision and Mathematics

in co-operation with

**EUROGRAPHICS**

# GraVisMa 2010

**Workshop Proceedings**

***Edited by***

**Vaclav Skala**, University of West Bohemia, Czech Republic
**Eckhard M. Hitzer**, University of Fukui, Japan

Václav Skala – UNION Agency

***GraVisMa 2010 Proceedings***

Editor-in-Chief: Vaclav Skala
c/o University of West Bohemia, Univerzitni 8
CZ 306 14 Plzen
Czech Republic
*skala@kiv.zcu.cz*

Managing Editor: Vaclav Skala

Published and printed by:
Vaclav Skala – Union Agency
Na Mazinách 9
CZ 322 00 Plzen
Czech Republic

Hardcopy: *ISBN 978-80-86943-85-5*

# Foreword

**Goals** of the GraVisMa workshops are to bring the theories of the Projective Geometry, Geometric Algebra and Conformal Geometry to practical application especially in the fields related to Computer Graphics and Vision, Scientific Computation and Visualization.

The 2nd International workshop on Computer Graphics, Computer Vision and Mathematics was held at Brno University of Technology at Brno, Czech Republic on September 7.- 10, 2010.

**GraVisMa** workshops are a unique forum of researchers, practitioners, developers and academic experts to discuss new approaches and methods in Computer Graphics, Computer Vision, Scientific Computation, Scientific, Medical and Information Visualization (and other relevant fields) with application of the latest developments in Mathematics and Physics and vice versa.

**GraVisMa** workshops bring new impulses related to computer science, especially in the development of new approaches and algorithms and stimulate research cooperations between mathematicians and computer science experts.

**GraVisMa** workshop proceedings contain full papers, communication papers and posters presented at the workshop that were commented, reviewed by participants and external reviewers. Papers accepted for publication were further improved after the workshop due to discussions and comments during the event.

# GraVisMa 2010 attendees

At the GraVisMa 2010 workshop were presented:

- two keynotes
  - Rockwood,A.: Feature based, transfinite interpolation with applications to industrial design and terrain modeling, Saudi Arabia & USA
  - Scheuermann,G.: New developments in mathematical visualization, Germany
- four tutorials
  - Tachibana,K., Hitzer,E.: Geometric algebra neural networks, Japan
  - Seybold,F.: Gaalet, Germany
  - Hildenbrand,D., Rockwood,A.: Introduction to (conformal) geometric algebra and its efficient implementation using Gaalop, Germany, Saudi Arabia & USA
  - Heidekrüger,A.: Introduction to OpenCL (AMD), Germany
- 23 research papers

Informal atmosphere of the GraVisMa 2010 workshop stimulated scientific discussions between researchers and practitioners that will hopefully lead to further research international research collaborations and common project proposals.

The co-chairs would like to thank to all:

- who contributed to this workshop, especially the reviewers,
- who helped so that this workshop could be held and to people that helped during the workshop,
- sponsoring organizations.

The next GraVisMa 2010 workshop will be held in the Ostrava city, which is very industrial with heavy industry and coal mines with many historical places. The workshop is planned for September 2011.

## Co-Chairs

Václav Skala
University of West Bohemia
Czech Republic

Eckhard M.Hitzer
University of Fukui
Japan

***GraVisMa* 2010 was supported by**:

University of West Bohemia Plzen
Brno University of Technology, Brno

**Microsoft** s.r.o.

**NIVIDIA**

Zoner Software a.s.

**AMD**
Advanced Micro Devices

three brothers production
www.historicalmovie.com

**SILICON GRAPHICS**
s.r.o

**Co-Chairs**

Eckhard M.S. Hitzer (Japan)
Vaclav Skala (Czech Republic)

**International Program Committee**

Ablamowicz,Rafal (United States)
Bayro-Corrochano,Eduardo (Mexico)
Dorst,Leo (Netherlands)
Gavrilova,Marina (Canada)
Groeller,Eduard (Austria)
Hildenbrand,Dietmar (Germany)
Hitzer,Eckhard (Japan)
Horn,Martin Erik (Germany)
Lasenby,Joan (United Kingdom)
Skala,Vaclav (Czech Republic)
Sojka,Eduard (Czech Republic)
Torrens,Francisco (Spain)
Zemcik,Pavel (Czech Republic)

**Organization Committee**

Hildebrand,Dietmar (Germany)
Skala,Vaclav (Czech Republic)
Zemcik,Pavel (Czech Republic)

**Members**

Herout,Adam (Czech Republic)
Kolcun,Alexej (Czech Republic)
Lavicka,Miroslav (Czech Republic)
Sojka,Eduard (Czech Republic)

**Board of Reviewers**

Ablamowicz,Rafal (United States)
Bayro-Corrochano,Eduardo (Mexico)
Bell,Ian (United Kingdom)
Benger,Werner (United States)
Cibura,Carsten (Netherlands)
De Floriani,Leila (Italy)
Deul,Crispin (Germany)
Dorst,Leo (Netherlands)
Fassold,Hannes (Austria)
Fuchs,Laurent (France)
Gain,James (South Africa)
Gavrilova,Marina (Canada)
Hasegawa,Makoto (Japan)
Havemann,Sven (Austria)
Herout,Adam (Czech Republic)
Hildenbrand,Dietmar (Germany)
Hitzer,Eckhard (Japan)
Horn,Martin Erik (Germany)
Isokawa,Teijiro (Japan)
Jolly,Raphael (France)
Jung,Yvonne (Germany)
Kapec,Peter (Slovakia)
Kohout,Josef (Czech Republic)
Kolcun,Alexej (Czech Republic)
Kooijman,Adrie (Netherlands)
Kortenkamp,Ulrich (Germany)
Lasenby,Anthony (United Kingdom)
Lasenby,Joan (United Kingdom)
Lavicka,Miroslav (Czech Republic)
Li,Hongbo (China)
Macdonald,Alan (USA)
Marais,Patrick (South Africa)
Michoud,Brice (France)
Panozzo,Daniele (Italy)
Puppo,Enrico (Italy)
Rocca,Luigi (Italy)
Rockwood,Alyn (Saudi Arabia)
Sabov,Alexander (Germany)
Saint-Jean,Christophe (France)
Sbert,Mateu (Spain)
Scheuermann,Gerik (Germany)
Schwinn,Christian (Germany)
Skala,Vaclav (Czech Republic)
Sojka,Eduard (Czech Republic)
Stork,Andre (Germany)
Tachibana,Kanta (Japan)
Torrens,Francisco (Spain)
Tytkowski,Krzysztof (Poland)
Vanecek,Petr (Czech Republic)
Vasa,Libor (Czech Republic)
Wiebel,Alexander (Germany)
Yuan,Linwang (China)
Zemcik,Pavel (Czech Republic)

# GraVisMa 2010

## Keynotes

Rockwood,A.: Feature based, transfinite interpolation with applications to industrial design and terrain modeling, Saudi Arabia & USA
Scheuermann,G.: New Developments in Mathematical Visualization, Germany

## Tutorials

Tachibana,K., Hitzer,E.: Geometric Algebra Neural Networks, Japan
Seybold,F.: Gaalet, Germany
Hildenbrand,D., Rockwood,A.: Introduction to (Conformal) Geometric Algebra and its efficient implementation using Gaalop, Germany, Saudi Arabia & USA
Heidekrüger,A.: Introduction to OpenCL (AMD), Germany

## Contents

## Plenary Papers

## Communication Papers



## Poster Papers

# Gaalop 2.0 - A Geometric Algebra Algorithm Compiler

Christian Schwinn
TU Darmstadt, Germany
Department of Computer Science
schwinn@rbg.informatik.tu-darmstadt.de

Dietmar Hildenbrand
TU Darmstadt, Germany
Department of Computer Science
Interactive Graphics Systems Group
dhilden@gris.informatik.tu-darmstadt.de

Florian Stock
TU Darmstadt, Germany
Department of Computer Science
Embedded Systems and Applications Group
stock@esa.informatik.tu-darmstadt.de

Andreas Koch
TU Darmstadt, Germany
Department of Computer Science
Embedded Systems and Applications Group
koch@esa.informatik.tu-darmstadt.de

**ABSTRACT**

In recent years, Geometric Algebra (GA) has become more and more popular in fields of science and engineering due to its potential for compact algorithms. However, the execution of GA algorithms and the related need for high computational power is still the limiting factor for these algorithms to be used in practice. Therefore, it would be desirable to automatically detect parts that can be calculated in parallel by a software tool. In this paper, we present Gaalop 2.0, a Geometric Algebra Algorithm Compiler, which takes as input the description of a GA algorithm, symbolically optimizes the output multivectors and compiles the optimized code into a target language source file such as C++, for instance. For each output multivector the code for the different coefficients is generated, which is finally adjusted to contain only basic arithmetic operations instead. This allows the optimized output to be compiled for parallel computing platforms like FPGAs, for instance.

**Keywords:** Geometric Algebra, Geometric Computing, Compiler, Optimization, FPGA, Parallel Computing.

## 1 INTRODUCTION

Geometric Algebra is a mathematical framework that facilitates the development of algorithms in different fields of engineering and research. Algorithms in GA are geometrically intuitive and very compact compared to conventional approaches. Examples for how Geometric Algebra can be applied in engineering or computer graphics are described in [4], [12], [7] or [15], for instance. A major drawback is the increased runtime, which is an implication of high dimensions in multivectors of geometric algebras ($2^n$ for dimension $n$). In order to make GA algorithms comparable to standard implementations, it is necessary to find optimizations that lead to a better runtime performance. Fortunately, the components of a multivector can be calculated in parallel. Implementing multivectors as a set of coefficients with associated basis blades makes it possible to find algebraic expressions for each coefficient separately. Blades are the basic geometric entities in geometric algebras. Multivectors consist of a linear combination of blades of different grades. Multivector coefficients can actually be calculated simultaneously, e.g. using parallel computing devices such as FPGAs. This paper presents Gaalop (Geometric Algebra Algorithm Optimizer), a compiler that calculates the very expressions for multivector components.



Gaalop optimizes Geometric Algebra algorithms written with the help of the CLUCalc software [13], using symbolic simplification backed by the Maple [10] Computer Algebra System (CAS), and compiles the output to different target languages, currently C/C++, CLUCalc, Verilog, DOT and LaTeX. The optimized code has no more Geometric Algebra operations and is ready to be run efficiently on various platforms, particularly on parallel computing architectures. This software is based on [8], a proof-of-concept implementation for high performance computing based on Conformal Geometric Algebra with a preliminary version of Gaalop.

This paper introduces Gaalop 2.0, a new version which has been completely rewritten in Java in order to extend the feature set of previous versions and to support multiple operating systems. Gaalop 2.0 incorporates multiple new features, primarily the support for control flow instructions and code generation for parallel computing platforms. We describe the optimization and compilation process performed by Gaalop 2.0.

This document is organized as follows: Section 2 shows related work in the field of GA implementations, Section 3 describes the input format based on CLUCalc. Section 4 gives an overview of the intermediate representation that is used between the compilation steps.

Section 5 shows the optimization process. Section 6 describes different code generators. Finally, Section 7 concludes this paper and gives an outlook to future work.

## 2 RELATED WORK

There are a lot of pure hardware or pure software solutions in order to optimize the performance of GA algorithms. The first FPGA (Field Programmable Gate Array) solution was described in [14], focusing only on geometric products. In [6], the CliffoSor solution was presented, implementing products with a restriction to 4-dimensional Geometric Algebras. [11] is the first solution without an integer restriction for coefficients. All of these pure hardware solutions are focusing on the implementation of general Geometric Algebra operations but do not use the high potential of symbolic optimizations. Pure software approaches are especially based on expression templates libraries (especially boost::math::clifford) or the Gaigen approach.

Gaigen [5] is a Geometric Algebra implementation generator that focuses on the generation of C++ code for specified algebra definitions. Given a signature and metric of a Geometric Algebra, Gaigen generates code that can be embedded into C++ applications directly. Generating code is a simple operation that does not require too much knowledge of Geometric Algebra features. However, implementations do not contain optimizations by default and thus suffer from performance lacks that can only be eliminated by applying optimizations manually. This requires to identify special cases in which multivectors can be reduced in size (called specializations), for instance geometric entities such as spheres which have only 1-dimensional blades (rather than bivectors, trivectors, etc.). Therefore, detailed knowledge is required from the user, since specializations cannot be deduced automatically. Gaalop does not require the step of manual optimization because multivectors are decomposed into their coefficients of basis blades.

## 3 INPUT FORMAT

Gaalop 2.0 supports a subset of CLUScript, a script language for the 3D visualization and scientific calculation software CLUCalc/CLUViz [13].With CLUCalc it is possible to develop algorithms visually, allowing rapid prototyping. This enables the user to create Geometric Algebra algorithms step by step, supported by visual output. Previous versions of Gaalop supported only sequential algorithms without conditional branches, loop statements or user-defined function calls. Table 1 compares the supported and new features in Gaalop 1.0 and 2.0. Algorithms to be optimized by Gaalop have to use Conformal Geometric Algebra only.

| | Version | |
|---|---|---|
| **CLUScript feature** | **1.0** | **2.0** |
| algebra definition | no | yes |
| pre-defined algebra functions | yes | yes |
| macros | no | yes |
| null-space definition | yes | yes |
| inner / outer / geometric products | yes | yes |
| if-statements | no | yes |
| loops | no | yes |
| variable lists / scopes / references | no | no |
| point operators | no | no |
| drawing / plotting functionality | no | no |
| LaTeX rendering | no | no |

**Table 1:** Comparsion of CLUScript support in Gaalop 1.0 and 2.0.

Restrictions to the full set of CLUScript features mainly concern visualization features, variable references or scopes. Note that the ? operator is interpreted in a slightly different way than in CLUScript. In Gaalop, this operator is used as a marker for lines of the input code for which the optimized output code should be generated. Therefore, this operator will be referenced to as the optimization marker throughout this document. Variables or lines that are not marked accordingly will be processed by Gaalop to find simplified expressions for multivector coefficients but not explicitly generated as output code. This makes it possible to optimize only crucial parts of an algorithm instead of each line which might not be important in the output.

### Example

```
DefVarsN3();
P = VecN3(px,py,pz);     // view point
M = VecN3(mx,my,mz);     // center point of earth
S = M-0.5*r*r*einf;      // sphere representing earth
K = P+(P.S)*einf;        // sphere around P
?C=S^K;                  // intersection circle
```

**Listing 1:** Sample input code showing a CLUScript file. Variables px, py, pz and mx, my, mz are free variables that will be handled symbolically.

Listing 1 shows a sample input script in Conformal Geometric Algebra. To illustrate the compilation process step by step, this code is taken as an example throughout this document. The task is to calculate an algebraic expression of the horizon on the earth viewed from an arbitrary point $P$. The earth is represented by a sphere $S$ with center $M$ and radius $r$ (line 3). Line 4 defines a sphere $K$ around $P$. The radius of this sphere is defined by the inner product of $P$ and $S$, which corresponds to the squared distance between $P$ and any point on $S$ that touches a tangent through $P$. Thus, $K$ has exactly the radius that matches the distance of the horizon

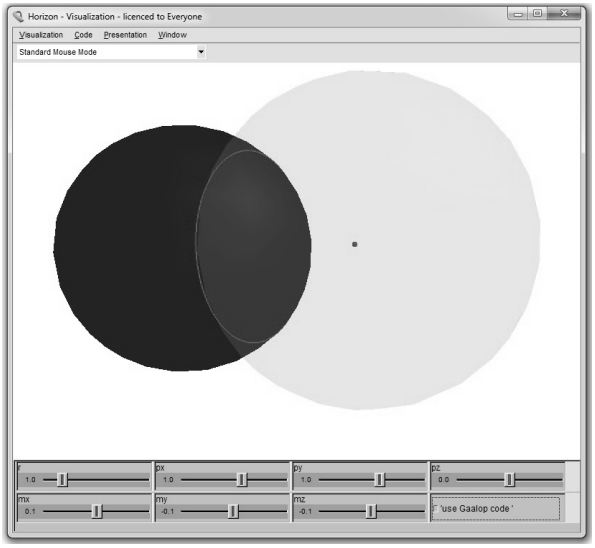



**Figure 1:** Screenshot of the CLUViz visualization window. Sliders can be used to modify input parameters. The check box on the bottom right allows to switch between original code and Gaalop optimized code. Errors in the optimized code would immediately become visible when switching modes.

to $P$. Finally, line 5 calculates the intersection circle $C$ modeling the horizon.

Figure 1 shows how the CLUViz visualization window looks like. The earth sphere $S$ is drawn in dark grey, the view point $P$ in shown on the right-hand side. The sphere $K$ around $P$ is indicated in light grey. The intersection circle is finally indicated between both spheres.

# 4 INTERMEDIATE REPRESENTATION

Gaalop parses input files and transforms the input into an intermediate representation, on which different compilation steps operate. For the parser implementation, the ANTLR parser generator tool has been used. Gaalop 2.0 uses two kinds of intermediate representations (IR), a control flow graph (CFG) and a data flow graph (DFG) to represent the algorithmic structure of the input program and related arithmetic operations such as assignments or application of mathematic functions, respectively. In fact, Gaalop builds a *control dataflow graph*, representing arithmetic expressions in terms of a DFG whose nodes themselves are referenced in CFG nodes. Hence, the DFG is not a graph of its own but rather implicitly contained in CFG nodes. Previous versions of Gaalop used a very simple type of control flow graph, which contained only sequential nodes, since real control flow like branches or loops was not supported.

## 4.1 Control Flow Graph

The control flow graph represents the overall structure of the input program. It distinguishes sequential statements such as assignments or procedure calls and control flow elements like if-statements or macros. As opposed to the data flow graph, the CFG does not represent details of arithmetic operations (e.g. additions, geometric products, etc.). Concrete types of CFG nodes are outlined below.

**Assignments** represent a variable to which an arbitrary expression is assigned. Both variable and expression are represented by an appropriate DFG node.

**Optimization markers** encapsulate a variable for which the optimized output code will be generated.

**If-statements** consist of a condition, a positive branch and a negative branch. Conditions are modeled via a DFG expression, branches as an (implicit) list of other CFG nodes. This type of node is self-contained, so nested statements are possible.

**Loops** contain the statements from the body, including termination conditions which are usually related to if-statements. Optionally, a number of iterations can be given which is used to unroll the loop.

**Macros** are represented by a name and associated list of statements. The name is further used to identify usages of this macro in the input code. This can be used to inline the use of macros in order to augment the range of optimization by replacing the call of a macro by its actual code.

Some CFG nodes contain references to arithmetic expressions which are related to the respective code in the input program. These expressions are modeled by the data flow graph.

**Example**

Figure 2 shows the control flow graph which corresponds to the input file from Listing 1. For each assignment there is an according node in the CFG. The optimization marker is represented by a dedicated node.

## 4.2 Data Flow Graph

The data flow graph represents the arithmetic parts of the input code. Elements of an assignment, such as variable and value, are modeled via DFG nodes. For each mathematical operation supported by Gaalop there exists a corresponding type of nodes, outlined below. The common basis of DFG nodes is an *expression* type. Concrete nodes can be placed on any location where an expression is expected.

**Unary operations** model operations like negation or dualization that take only one expression as argument.

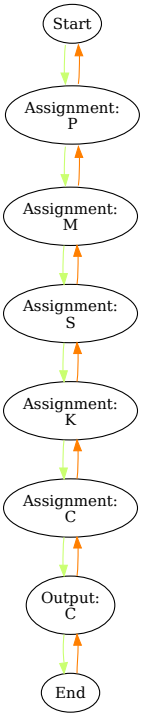



**Figure 2:** Control flow graph corresponding to the input file from section 3. The algebra definition from the input code is handled separately. Start and end node are special marker nodes.

**Binary operations** model operations like addition, subtraction or inner / outer / geometric products which take two expressions as argument. Each binary operation has a left and right operand.

**Language elements** consist of pre-defined function calls (e.g. `VecN3` to define a conformal point) or mathematic functions like $sin$, $cos$ or $sqrt$. These functions take different numbers and types of arguments, each of which is another expression.

**Identifiers** are the actual parameters of functions, operations and relations. These can be variables, integer or float constants and basis vectors.

Other CLUCalc relevant language elements such as null space definitions or algebra selection are not modeled as dedicated DFG nodes. These are handled by a separate type which will be referenced to as *algebra signature*. This signature is directly referenced by the control flow graph, since contained properties are global to the input code and not related to single CFG or DFG nodes.

Gaalop supports Conformal Geometric Algebra, defined by the `DefVarsN3` function in CLUScript. This algebra has an associated signature and blade list, which defines the elementary basis blades. Table 2 lists the 32 basis blades of the Conformal Geometric Algebra in the canonical ordering. This table can also be seen as a lookup table for the association between multivector coefficient and the related blade, as it is used by the code generators (Section 6).

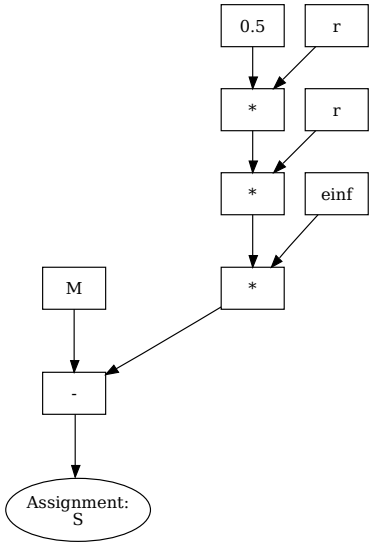



**Figure 3:** Data flow graph corresponding to the assignment to variable $S$ in the input file from section 3.

### Example

Figure 3 shows the data flow graph which corresponds to the assignment to variable $S$ in the input file from Listing 1.

## 5 OPTIMIZATION

Gaalop 2.0 has been re-designed to have a strictly modular interface. In previous versions, different parts of the program such as parser, optimizer and code generator have been hardwired and were neither exchangeable or extensible. In Gaalop 2.0, these parts are modeled as plugins which can easily be exchanged and extended without modifying the main program. Especially the intermediate representation has been separated to be accessible from all modules that read, modify or write this structure.

The compilation process from input file to the optimized output code consists of three major passes. Starting with the input file, Gaalop parses this file to produce the intermediate representation (CFG / DFG), optimizes the input by symbolic manipulation and generates the output code depending on the selected target language. These passes are explained below.

The optimizer is responsible for symbolic simplification and calculation of optimized multivector coefficients for expressions marked with the optimization marker (`?`) in the input code. This compilation pass is implemented using the OpenMaple interface of the Maple Computer Algebra System (CAS) [10] with the CLIFFORD library by Rafal Ablamowicz [1] for Geometric Algebra calculations. This allows

| index | blade | grade |
|---|---|---|
| **0** | $1$ | 0 |
| **1** | $e_1$ | 1 |
| **2** | $e_2$ | 1 |
| **3** | $e_3$ | 1 |
| **4** | $e_\infty$ | 1 |
| **5** | $e_0$ | 1 |
| **6** | $e_1 \wedge e_2$ | 2 |
| **7** | $e_1 \wedge e_3$ | 2 |
| **8** | $e_1 \wedge e_\infty$ | 2 |
| **9** | $e_1 \wedge e_0$ | 2 |
| **10** | $e_2 \wedge e_3$ | 2 |
| **11** | $e_2 \wedge e_\infty$ | 2 |
| **12** | $e_2 \wedge e_0$ | 2 |
| **13** | $e_3 \wedge e_\infty$ | 2 |
| **14** | $e_3 \wedge e_0$ | 2 |
| **15** | $e_\infty \wedge e_0$ | 2 |
| **16** | $e_1 \wedge e_2 \wedge e_3$ | 3 |
| **17** | $e_1 \wedge e_2 \wedge e_\infty$ | 3 |
| **18** | $e_1 \wedge e_2 \wedge e_0$ | 3 |
| **19** | $e_1 \wedge e_3 \wedge e_\infty$ | 3 |
| **20** | $e_1 \wedge e_3 \wedge e_0$ | 3 |
| **21** | $e_1 \wedge e_\infty \wedge e_0$ | 3 |
| **22** | $e_2 \wedge e_3 \wedge e_\infty$ | 3 |
| **23** | $e_2 \wedge e_3 \wedge e_0$ | 3 |
| **24** | $e_2 \wedge e_\infty \wedge e_0$ | 3 |
| **25** | $e_3 \wedge e_\infty \wedge e_0$ | 3 |
| **26** | $e_1 \wedge e_2 \wedge e_3 \wedge e_\infty$ | 4 |
| **27** | $e_1 \wedge e_2 \wedge e_3 \wedge e_0$ | 4 |
| **28** | $e_1 \wedge e_2 \wedge e_\infty \wedge e_0$ | 4 |
| **29** | $e_1 \wedge e_3 \wedge e_\infty \wedge e_0$ | 4 |
| **30** | $e_2 \wedge e_3 \wedge e_\infty \wedge e_0$ | 4 |
| **31** | $e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$ | 5 |

**Table 2:** Blades of the 5D conformal geometric algebra and their ordering.

to use Maple for symbolic calculations from Java directly. The Maple optimizer traverses the input control dataflow graph as it has been set up by the parser in the previous compilation pass (Figures 2, 3). On each optimization step, parts of the control or data flow graph are removed or replaced by expressions representing simplified calculations. After the optimization pass, the CFG and DFG contain only optimized code. Assignments which have not been marked for optimization are not contained in the graph anymore. Gaalop optimizes the input trying to achieve two objectives:

- Symbolic simplification. Each assignment of the input script is translated to Maple syntax and sent to the Maple engine. Maple keeps track of commands executed by the engine and symbolically simplifies assignments where appropriate. The original assignments from the input CFG are removed and replaced in favor of new assignments, which calculate the individual multivector coefficients using scalar arithmetic operations only.
- Preparation for parallel computing platforms. Multivector components, i.e. coefficients of a multivector's linear combination of basis blades, can be calculated in parallel. This holds potential for parallel execution of instructions calculating the non-zero coefficients. For an overview of the basis blades of the Conformal Geometric Algebra, refer to Table 2.

To reach these goals, CFG nodes are translated to Maple syntax and sent to the Maple engine. The concrete representation is a string value describing the command to be executed by Maple. Therefore, control flow nodes are processed in the following way.

**Assignments** are translated according to the syntax rules of the CLIFFORDLIB package for Maple, e.g. using the `&c` operator for the geometric product.

**Optimization markers** trigger the calculation of simplified multivector coefficients for the selected variable. Thus, a self-defined Maple procedure decomposes the multivector to its $2^n$ components. For each component, the relevant part of the linear combination of basis blades is selected. Finally, the related coefficient is evaluated and symbolically simplified. The resulting multivector is put into an array of multivector components which is used to return the expression into the control flow graph. Before returning control to Gaalop, another procedure tries to replace variable references that have previously been optimized, so results that have already been calculated can be reused in further calculations. Afterwards, Gaalop removes assignments to the old multivector and inserts new assignments for each non-zero multivector component. Each of these assignments represents the simplified expression that calculates the coefficient of the respective basis blade (Table 2).

**If-statements** are processed recursively by traversing the instructions of their positive and negative branches. In both branches, each line is optimized automatically in order to eliminate Geometric Algebra operations. This is necessary for the use of backends which do not support Geometric Algebra.

**Macros** are inlined in a separate pass. For each macro call, the corresponding code is copied and the call is replaced by the actual return value.

**Loops** can be processed in two ways: either being unrolled by a fixed number of iterations or similar to if-statements to eliminate Geometric Algebra operations. In each case, further information is required which can be given in the input file using dedicated `#pragma` comments. If the number of iterations for the loop is known, the loop can be unrolled by copying the body n times. In the normal case where only Geometric Algebra operations are removed, control

variables have to be given in order to correctly process statements in the body. Due to the generalized CLUCalc syntax for loops, which does not specify the concrete type of loop, Gaalop would otherwise remove assignments to control variables like counter variables, breaking termination conditions in the output.

Listing 2 shows a simple example how a loop can be defined in the input file to be unrolled in Gaalop. The loop body is copied 10 times before the optimizer processes assignments, making the original loop disappear in the output. In the best case, the entire loop can be reduced to a single statement.

```
p = VecN3(x,y,z);
i = 0;
//#pragma unroll 10
loop {
   if (i > 9) {
      break;
   }
   // do something
}
?p;
```

**Listing 2:** Simple code fragment showing the use of unrolling loops.

After this transformation, assignment nodes are replaced by the simplified expressions that have been calculated by Maple and inserted into the CFG. Hence, the CFG has been modified to contain the optimized code instead.

### Example

The modified control dataflow graph corresponding to the example from Listing 1 is too large to be illustrated here. Nevertheless, it is easy to imagine the general structure of the graph after the optimization pass. Remember that only one variable has been marked by ? to be printed in the optimized output (the intersection circle $C$). Hence, the assignments to other variables have been removed without replacement (their contribution to the result is implicitly contained in the value of $C$). As the assignment to $C$ has been replaced by the assignments to its multivector components, there are 10 new nodes now, representing the assignments to the bivector parts of the multivector (indices 6-15, see Table 2). At the end of the new graph there is an output (optimization) node representing the overall multivector $C$.

## 6 CODE GENERATOR

After the optimization pass, the intermediate representation contains the simplified expressions calculating the result multivectors that were marked in the input file. The IR is now ready to be processed by code generators, also called backends of the compiler. Each backend is implemented as a plugin to Gaalop which can be selected before the compilation process starts.

Code generators traverse the IR to translate the optimized code to the syntax rules of the target platform. For backends that do not support parallel computing, no modifications to the IR have to be performed. This is the case for most backends implemented in Gaalop 2.0 such as CLUCalc, C++, DOT or LaTeX. The more advanced Verilog backend performs additional manipulations to the IR in order to prepare the output for parallel architectures such as an FPGA.

A common property of all backends is that no Geometric Algebra module has to be included. Since the multivector descriptions have been optimized to expressions containing only conventional scalar arithmetic operators, no time-consuming operations, such as the geometric product, have to be executed. Generated output always operates on lists or arrays of coefficients, which are directly related to the blade indices in the canonical ordering of Table 2, rather than on the geometric entities (blades) themselves.

### CLUCalc

Even if CLUCalc is the input language, it is also sensible to offer it as backend, too. In this manner, the correctness of the Gaalop compiler can be verified by comparing the original and optimized code visually in the CLUViz software (cf. Figure 1).

```
DefVarsN3();
C_opt = List(32);
C_opt(7) = -(my * px) + mx * py; // e1^e2
C_opt(8) = -(mz * px) + mx * pz; // e1^e3
C_opt(9) = (((mx * pz * mz + mx * py * my) - 0.5 *
     mx^^3.0 + 0.5 * mx^^2.0 * px) - 0.5 * mx * my
     ^^2.0 - 0.5 * mx * mz^^2.0 + 0.5 * mx * r^^2.0)
      - 0.5 * px * my^^2.0 - 0.5 * px * mz^^2.0 +
     0.5 * px * r^^2.0; // e1^einf
C_opt(10) = -(1.0 * px) + mx; // e1^e0
C_opt(11) = -(mz * py) + my * pz; // e2^e3
C_opt(12) = (((my * px * mx + my * pz * mz) - 0.5 *
     my^^3.0 - 0.5 * my * mx^^2.0 + 0.5 * my^^2.0 *
     py) - 0.5 * my * mz^^2.0 + 0.5 * my * r^^2.0) -
      0.5 * py * mz^^2.0 - 0.5 * py * mx^^2.0 + 0.5
     * py * r^^2.0; // e2^einf
C_opt(13) = -(1.0 * py) + my; // e2^e0
C_opt(14) = (((mz * px * mx + mz * py * my) - 0.5 *
     mz^^3.0 - 0.5 * mz * mx^^2.0 + 0.5 * mz^^2.0 *
     pz) - 0.5 * mz * my^^2.0 + 0.5 * mz * r^^2.0) -
      0.5 * pz * my^^2.0 - 0.5 * pz * mx^^2.0 + 0.5
     * pz * r^^2.0; // e3^einf
C_opt(15) = -(1.0 * pz) + mz; // e3^e0
C_opt(16) = ((-(1.0 * px * mx) - 1.0 * py * my + my
     ^^2.0 + mz^^2.0) - 1.0 * r^^2.0 + mx^^2.0) -
     1.0 * pz * mz; // einf^e0
?C = C_opt(7) * e1^e2 + C_opt(8) * e1^e3 + C_opt(9)
     * e1^einf + C_opt(10) * e1^e0 + C_opt(11) * e2^
     e3 + C_opt(12) * e2^einf + C_opt(13) * e2^e0 +
     C_opt(14) * e3^einf + C_opt(15) * e3^e0 + C_opt
     (16) * einf^e0;
```

**Listing 3:** Optimized CLUCalc output for the input file from Listing 1 in section 1. Multivector components and associated blades are indexed according to Table 2 incremented by 1, since counting of list elements in CLUCalc starts with 1.

As a concrete example, listing 3 shows the resulting CLUCalc output code for the input file from section 1.

C is defined as a list of 32 entries with coefficients 7 to 16 set to the optimized expressions as calculated from the optimizer, while other coefficients are zero. The associated blade indices correspond directly to the ones defined in Table 2, incremented by 1 to match the 1-based counting of list elements in CLUCalc. In the last line, C is reassembled using the coefficients and their associated basis blades.

### C/C++

The C/C++ backend optionally wraps the generated code in a `calculate` method that takes the unknown input variables and a reference to the output multivector as parameters. Multivectors are handled as float arrays whose indices correspond to the ones from Table 2. Code generated from this backend can be used in an existing C++ program. Passing the correct parameters to the calculate function, the result can be calculated without knowledge of GA operations.

Listing 4 shows the resulting C/C++ output code for the input file from section 1. C is defined as an array of float with 32 entries.

```
void calculate(float mx, float my, float mz, float
    px, float py, float pz, float r, float C[32]) {
  C[6] = mx * py - my * px; // e1^e2
  C[7] = -(mz * px) + mx * pz; // e1^e3
  C[8] = mx * py * my + mx * pz * mz - 0.5f * pow(
       mx,3.0f) + 0.5f * mx * mx * px - 0.5f * mx *
        my * my - 0.5f * mx * mz * mz + 0.5f * mx *
        r * r - 0.5f * px * my * my - 0.5f * px *
       mz * mz + 0.5f * px * r * r; // e1^einf
  C[9] = -(1.0f * px) + mx; // e1^e0
  C[10] = -(mz * py) + my * pz; // e2^e3
  C[11] = my * px * mx + my * pz * mz - 0.5f * pow(
       my,3.0f) + 0.5f * my * my * py - 0.5f * my *
        mx * mx - 0.5f * my * mz * mz + 0.5f * my *
        r * r - 0.5f * py * mz * mz - 0.5f * py *
       mx * mx + 0.5f * py * r * r; // e2^einf
  C[12] = -(1.0f * py) + my; // e2^e0
  C[13] = mz * py * my + mz * px * mx - 0.5f * pow(
       mz,3.0f) + 0.5f * mz * mz * pz - 0.5f * mz *
        mx * mx - 0.5f * mz * my * my + 0.5f * mz *
        r * r - 0.5f * pz * my * my - 0.5f * pz *
       mx * mx + 0.5f * pz * r * r; // e3^einf
  C[14] = -(1.0f * pz) + mz; // e3^e0
  C[15] = -(1.0f * px * mx) - 1.0f * py * my - 1.0f
        * pz * mz + mz * mz - 1.0f * r * r + mx *
       mx + my * my; // einf^e0
}
```

**Listing 4:** Optimized C/C++ output for the input file from Listing 1 in section 1. Multivector components and associated blades are numbered according to Table 2.

### DOT

The DOT backend generates a .dot file for visualization with the Graphviz Graph Visualization Software [2]. This is helpful to inspect the intermediate representation as it has been modified by the optimizer. For example, Figures 2 and 3 show parts of the input file's IR which was generated by the DOT code generator with the Maple optimization disabled.

### LaTeX

For scientific reports about algorithms optimized with Gaalop, it has been necessary to manually transform the output code to a human-readable text. The LaTeX backend automates this step by generating a description of the output code in form of math formulae that can be embedded in a .tex document. The equations from Figure 4 have been generated by Gaalop 2.0 according to the example from Listing 1.

$$
\begin{aligned}
C_6 &= mx * py - my * px \\
C_7 &= -mz * px + mx * pz \\
C_8 &= -\frac{1}{2}mx * mz^2 - \frac{1}{2}mx * my^2 + \frac{1}{2}mx * r^2 \\
&\quad + \frac{1}{2}mx^2 * px - \frac{1}{2}px * my^2 - \frac{1}{2}px * mz^2 \\
&\quad + \frac{1}{2}px * r^2 - \frac{1}{2}mx^3 \\
&\quad + mx * py * my + mx * pz * mz \\
C_9 &= -1 * px + mx \\
C_{10} &= -mz * py + my * pz \\
C_{11} &= \frac{1}{2}my^2 * py - \frac{1}{2}my * mz^2 - \frac{1}{2}my * mx^2 \\
&\quad + \frac{1}{2}my * r^2 - \frac{1}{2}py * mz^2 - \frac{1}{2}py * mx^2 \\
&\quad + \frac{1}{2}py * r^2 - \frac{1}{2}my^3 \\
&\quad + my * pz * mz + my * px * mx \\
C_{12} &= -1 * py + my \\
C_{13} &= \frac{1}{2}mz^2 * pz - \frac{1}{2}mz * mx^2 - \frac{1}{2}mz * my^2 \\
&\quad + \frac{1}{2}mz * r^2 - \frac{1}{2}pz * my^2 - \frac{1}{2}pz * mx^2 \\
&\quad + \frac{1}{2}pz * r^2 - \frac{1}{2}mz^3 \\
&\quad + mz * py * my + mz * px * mx \\
C_{14} &= -1 * pz + mz \\
C_{15} &= -1 * py * my - 1 * px * mx - 1 * pz * mz \\
&\quad + mz^2 - 1 * r^2 + mx^2 + my^2
\end{aligned}
$$

**Figure 4:** LaTeX representation of the optimized output for the input file from Listing 1 in section 1. The actual LaTeX code is wrapped into an align environment.

### Verilog

The Verilog Hardware Description Language (HDL) is used to describe hardware circuits. The Verilog backend produces synthesizeable Verilog HDL code, i.e. it can be used for the production of an application specific circuit (ASIC) or it can be mapped on a reconfigurable unit, like a field programmable gate array (FPGA). Similar to the C/C++ backend an independent module is generated, which does not rely on a specific architecture or infrastructure.

As the generated hardware is fully spatial parallel and pipelined, it consumes in each cycle a date and returns a computation result. The latency of the computation depends on the input program and the involved operations.

To generate the hardware, the backend transforms the control flow graph into a pure data flow graph. As more complex control flow like loops is currently not yet supported in this backend, this is always possible. The nodes of the newly generated data flow graph consist only of operators which can be mapped to Verilog via a special library.

Before the final mapping to the operations is done, some optimization is applied to reduce the required silicon area: Constant Propagation, Constant Folding, Common Subexpression Elimination, and in case the user selected not to implement the computation as floating point but as fixed point a Monte-Carlo-Simulation for word length optimization.

Optimization continues while mapping hardware. Multiplications and division by a power of two can be replaced in hardware with a new wiring, and each operator is instantiated with the right size to minimize area requirements.

## 7 CONCLUSION & FUTURE WORK

We presented Gaalop 2.0, an advanced version of the Gaalop compiler. Written in Java, Gaalop 2.0 can be used on any platform where Maple is installed. We have introduced CLUScript as the input language for algorithms to be optimized with Gaalop, giving an overview of supported language features. After introducing the intermediate representation for the internal handling of the input code, we focused on the optimization process, giving details about the input parser, Maple simplifier and different code generators. We showed how Gaalop 2.0 transforms the input code to an optimized representation that goes without explicit Geometric Algebra operations. Exploiting the fact that multivector components can be calculated simultaneously, we have implemented a Verilog code generator which produces code that compiles the input algorithm to an FPGA hardware description.

We plan to extend the set of code generators by backends for general-purpose computing platforms. Different standards for multicore architectures like OpenMP (Open Multi-Processing) [3] or OpenCL (Open Computing Language) [9] offer the opportunity to make use of the huge processing power of modern computers. This is where two worlds come together: High-dimensional Geometric Algebras offering an elegant and intuitive way of describing algorithms, requiring considerable computing performance, and multicore architectures taking advantage of the increasing parallelism in integrated circuits as compensation of lacking performance of GA algorithms. The combination of these approaches advances to the vision of a "Geometric Algebra Computer" that accelerates standard implementations while keeping algorithms compact and intuitive.

## REFERENCES


[1] Rafal Ablamowicz. Clifford algebra computations with Maple. In W. E. Baylis, editor, *Clifford (Geometric) Algebras*, pages 463–501. Birkhäuser, Boston, 1996.

[2] AT&T and Bell-Labs. Graphviz - Graph Visualization Software. http://www.graphviz.org/.

[3] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP : portable shared memory parallel programming*. The MIT Press, 2008.

[4] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufman, 2007.

[5] Daniel Fontijne. Gaigen 2: A Geometric Algebra Implementation Generator. In *GPCE'06*, http://staff.science.uva.nl/ fontijne/gaigen2.html, 2006. ACM.

[6] Antonio Gentile, Salvatore Segreto, Filippo Sorbello, Giorgio Vassallo, Salvatore Vitabile, and Vincenzo Vullo. CliffoSor, an Innovative FPGA-based Architecture for Geometric Algebra. In *ERSA 2005*, pages 211–217, 2005.

[7] Dietmar Hildenbrand, Daniel Fontijne, Christian Perwass, and Leo Dorst. Tutorial geometric algebra and its application to computer graphics. In *Eurographics conference Grenoble*, 2004.

[8] Dietmar Hildenbrand, Joachim Pitt, and Andreas Koch. *Geometric Algebra Computing for Engineering and Computer Science*, volume 1, chapter Gaalop - High Performance Parallel Computing based on Conformal Geometric Algebra, pages 350–358. Springer, 2010.

[9] Khronos-Group. The OpenCL home page. Available at http://www.khronos.org/opencl/, 2009.

[10] Maplesoft. OpenMaple - an API into Maple. http://www.maplesoft.com/applications/view.aspx?SID=4383.

[11] Biswajit Mishra and Peter R. Wilson. VLSI implementation of a geometric algebra parallel processing core. Technical report, Electronic Systems Design Group, University of Southampton, UK, 2006.

[12] Christian Perwass. *Geometric Algebra with Applications in Engineering*. Springer, 2009.

[13] Christian Perwass. CLUCalc / CLUViz Interactive Visualization. http://www.clucalc.info, 2010.

[14] Christian Perwass, Christian Gebken, and Gerald Sommer. Implementation of a Clifford algebra co-processor design on a field programmable gate array. In Rafal Ablamowicz, editor, *CLIFFORD ALGEBRAS: Application to Mathematics, Physics, and Engineering*, Progress in Mathematical Physics, pages 561–575. 6th Int. Conf. on Clifford Algebras and Applications, Cookeville, TN, Birkhäuser, Boston, 2003.

[15] John A. Vince. *Geometric Algebra for Computer Graphics*. Springer, 2008.

# Efficient Multi-scale Curvature and Crease Estimation

D. Panozzo
DISI - Università di Genova
panozzo@disi.unige.it

E. Puppo
DISI - Università di Genova
puppo@disi.unige.it

L. Rocca
DISI - Università di Genova
rocca@disi.unige.it

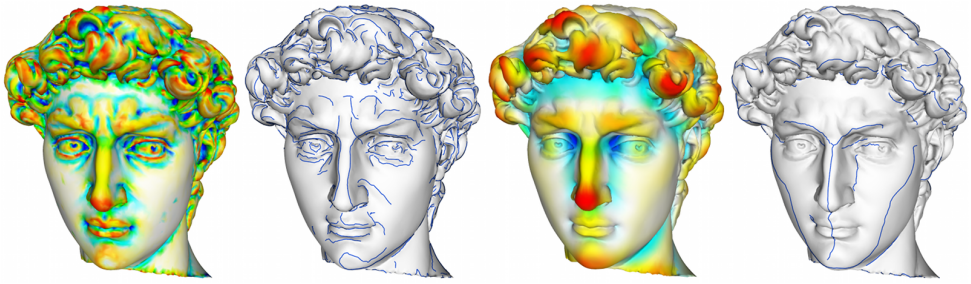


**Figure 1:** *Principal curvatures and creases computed at scales* $2e$ *and* $10e$*, with* $e$ *the average length of mesh edges. Color map combines principal curvatures with a non-linear mapping designed to enhance small variations: red convex; yellow flat-convex; green saddle; cyan flat-concave; blue concave; white flat. At the large scale, the two ridges along the nose merge into one ridge that extends downwards to the chin; transverse creases along lips and eyelids disappear; and a new crease along the convex ridge through cheekbones-cheeks-chin appears.*

## Abstract

We consider the problem of multi-scale estimation of principal curvatures and crease lines on a surface represented with a mesh of triangles and affected by noise. We show that curvature at different scales can be efficiently and accurately estimated by modifying a fitting technique and applying it to neighborhoods of various size, depending on scale: we discard bending portions of surfaces during fitting, and we apply Monte-Carlo sampling to speed up computation. Next we propose a new technique for extracting crease lines and we show how such lines can summarize shape features at the various scales. This is a first step towards building a scale-space of surface features.

**Keywords:** geometric mesh, curvature estimation, crease estimation, multi-scale surface analysis

## 1 Introduction

Several problems in computer graphics, geometric modeling and engineering involve the computation of differential properties of surfaces that are smooth in principle, while most often they are approximated with polygonal meshes. This subject has been treated by many authors in different contexts during the last twenty years, and several algorithms have been proposed for computing differential properties on geometric meshes or clouds of points. Many real datasets, e.g., those generated from range scanning, are affected by noise. Processing noisy meshes to estimate the differential properties of surfaces they represent can be a very challenging task, since the effect of noise is highly amplified by differentiation. This fact drastically restricts the range of applicability of most methods for differential surface analysis.



The problem of noise can be addressed in a more general way by multi-scale analysis, which has been successfully and extensively applied in image processing and computer vision [Lindberg 2009; Koenderink 1994]. Scale is a parameter to be used while analyzing the surface, which implicitly introduces a smoothing effect: multi-scale differential analysis of a surface is in fact analogous to differential analysis of different versions of the surface smoothed with filters at various scales.

In this framework, both noise and true shape features that are too small for the purpose of a given application can be filtered out. The finest relevant scale for a given representation cannot be finer than the scale $\sigma_n$ of noise. Therefore, if the surface is analyzed at a scale smaller than $\sigma_n$, estimates shall be unreliable; if it is analyzed at a scale $\sigma > \sigma_n$, estimates should be robust to noise, while also disregarding surface features at scales smaller than $\sigma$.

Multi-scale surface analysis may eventually lead to the construction of a *scale-space* of surfaces [Pauly et al. 2006]. Careful scale-space analysis may provide scale-persistent features to be used for diverse purposes such as surface simplification and remeshing, surface description and retrieval, non-photorealistic rendering, etc.

In [Yang et al. 2006; Pottmann et al. 2007], Pottmann et al. proposed a method for evaluating the curvature of a mesh at different scales without prior smoothing the mesh, but simply considering point neighborhoods of various sizes to perform integral computations. In line with their work, we present here a different approach to multi-scale estimation of curvatures, and we extend it to multi-scale extraction of creases (i.e., third order differential properties).

The contribution of our work is threefold. First, we show how a fitting technique for curvature estimation can be made efficient, accurate and robust to noise, and we show its application to multi-scale surface analysis. Second, we propose a new discrete and efficient method for estimating multi-scale creases, which exploits the results obtained during curvature estimation. Finally, we show the behavior of creases

extracted at the various scales, thus paving the way for the construction of a scale-space of surface features. Extensive experiments are presented to test our methods on real data and to compare them with existing methods.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we give the necessary background. In Section 4 we describe the method for curvature estimation, while in Section 5 we describe the method for crease extraction. In Section 6 we present experimental results. Finally, in Section 7 we make some concluding remarks.

## 2 Related work

The literature on computing curvatures and other differential quantities on surfaces and meshes is huge and many different methods have been developed in engineering, geometric modeling and computer graphics. Some recent accounts on the subject can be found, e.g., in [Costa Batagelo and Wu 2007; Gatzke and Grimm 2006; Grinspun et al. 2006]. In the following we briefly discuss just issues directly related to multi-scale estimation of differential quantities on surfaces potentially affected by noise.

Methods for estimating surface curvature can be broadly divided in two categories: *fitting methods* that fit analytic surfaces to data and derive differential properties analytically; and *discrete methods* that are all based on concepts of discrete differential geometry [Desbrun et al. 2005].

Discrete methods are usually fast and accurate in capturing the local surface properties, but most of them are very sensitive to noise. Moreover, most such methods are based on very local information (e.g., from the 1-ring of each vertex) and they can be hardly extended to a multi-scale approach. Notable exceptions are: the *Integral invariant* method proposed by Pottmann et al. [Pottmann et al. 2007; Yang et al. 2006]; the *Normal cycles* method proposed by Cohen-Steiner and Morvan [Cohen-Steiner and Morvan 2003]; and the method based on adaptive curve sampling proposed by Agam and Tang [Agam and Tang 2005]. The former method has been designed specifically for multi-scale computation, while the latter two can be easily extended to the purpose.

Fitting methods are overall more robust to noise and they naturally extend to multi-scale computation by considering larger neighborhoods for fitting. On the other hand, they are usually slower than discrete methods, and they introduce smoothing effects even at small scales. Moreover, they can become very inaccurate if fitting assumptions (e.g., on the projectability of the surface to a reference plane) are violated.

Different fitting methods are characterized by the type and degree of functions that they fit and by the information they use. Many methods assume that the surface normal is either available, or reliably estimated during pre-processing, and they fit a polynomial defined with respect to the tangent plane at the surface in a given point. One notable example is the method proposed by Goldfeather and Interrante [Goldfeather and Interrante 2004], which use cubic polynomials to fit both the position and the surface normal of data from the 1-ring of each vertex. As reported in [Gatzke and Grimm 2006], this method is very accurate if the surface normal is computed analitically, but it is very sensitive to error in estimating the normals. Cazals and Pouget [Cazals and Pouget 2005a] show that in fact accuracy of the estimated normal may have a high influence on the computation of higher order differential quantities. Thus, they propose the *Osculating Jets* method [Cazals and Pouget 2005a; Cazals and Pouget 2008], which fits polynomials of arbitrary degree, defined on a reference plane that goes through the given point, but is not necessarily tangent to the surface. They show the convergence of the method for analytic surfaces. Our method for estimating the normal direction and curvature tensor is a variation of the Osculating Jets.

Douros and Buxton [Douros and Buxton 2002] fit implicit functions locally to data and derive differential properties analytically from them. This approach could be extended to a multi-scale by using neighborhoods of various size, but it is computationally more involved - thus less efficient - than the Osculating Jets. Ohtake et al. [Ohtake et al. 2004] fit an implicit function to the whole dataset and derive differential properties analytically from it. Since the implicit function is generated from a sequence of approximations of the original data at different scales, this method can also be extended to a multi-scale one. On the other hand, generating the implicit function is equivalent to resolving a problem of surface reconstruction from a point cloud, which is far more complicated than the problem of estimating differential quantities.

Less works have been proposed to evaluate higher order differential quantities, such as creases. Ohtake et al. [Ohtake et al. 2004] evaluate extremalities (i.e., curvature derivatives, which are third order differential quantities - see Section 3) analytically from the implicit function they fit to data, then they extract the intersections of ridges with mesh edges by linear interpolation. They also propose a simple technique for filtering spurious ridges on the basis of the integral of curvature along each ridge. We adopt their filter in our work. Yoshizawa et al. [Yoshizawa et al. 2005] use the cubic fit technique of [Goldfeather and Interrante 2004] to estimate extremalities analytically, then they use variations of the method in [Ohtake et al. 2004] to extract and filter ridges. Hildebrandt et al. [Hildebrandt et al. 2005] use discrete and very local methods to evaluate all differential quantities. Once curvatures and curvature directions have been computed, they estimate extremalities on each triangle of the mesh by linear interpolation, and they extend such extremalities to each vertex by averaging values obtained at its incident triangles. Laplacian smoothing is performed on the resulting piecewise-linear scalar field, and creases are extracted triangle by triangle by linear interpolation.

Cazals and Pouget [Cazals and Pouget 2008] compute third and fourth order differential quantities necessary to compute extremalities analytically from a polynomial of order four (at least) fitted to data via the Osculating Jets. Once such coefficients have been computed, ridges are extracted with an algorithm described in [Cazals and Pouget 2005b], which also detects umbilical points and correctly manages ridges in their vicinity.

We are not aware of any work on extracting creases and higher order differential quantities at multiple scales.

## 3 Background

We summarize basic notions of differential geometry, which can be found in detail in any textbook, e.g., [Porteous 2001].

Let $\mathcal{S}$ be a smooth surface and let $N_{\mathcal{S}} : \mathcal{S} \rightarrow \mathbb{R}^3$ be its normal field (aka *Gauss map*), i.e., the field associating to each point $P \in \mathcal{S}$ its surface normal $N_{\mathcal{S}}(P)$. The *shape*

*operator* is the negative differential of the Gauss map, i.e.:

$$S = -dN_{\mathcal{S}}$$

that associates to each point $P \in \mathcal{S}$ a linear operator describing how the normal vector changes along any direction on the tangent plane of $\mathcal{S}$ at $P$. The shape operator is a tensor which can be described at each point $P$ by a $2 \times 2$ matrix $S_P$, referred to a local orthonormal frame $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ having its origin at $P$ and $\mathbf{n} = N_{\mathcal{S}}(P)$. Vectors $\mathbf{u}$ and $\mathbf{v}$ are a basis of the tangent space $T(P)$ at $P$. The eigenvalues $k_1$ and $k_2$ and eigenvectors $\mathbf{t_1}$ and $\mathbf{t_2}$ of matrix $S_P$ (in the local frame) define the values and directions of the *principal curvatures* of $\mathcal{S}$ at $P$, respectively. The principal directions of curvature are mutually orthogonal and lie on the tangent plane $T(P)$. Note that principal curvatures define line fields, rather than vector fields, on the surfaces of $\mathcal{S}$, therefore the orientations of $\mathbf{t_1}$ and $\mathbf{t_2}$ are defined arbitrarily. Hereafter we will assume $k_1 \geq k_2$ and we will select an orientation for curvature directions such that $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$ form a right-handed coordinate system, called the *Monge frame*.

Given $P \in \mathcal{S}$, the surface in a neighborhood of $P$ can be expressed in parametric form as $\mathbf{X}(u, v)$ with $(u, v) \in \Omega \subseteq \mathbb{R}^2$. In this case, the shape operator at $P$ can be described in terms of the first and second fundamental forms of $\mathbf{X}$.

The *first fundamental form* is the inner product on the tangent space $T(P)$: let $P = \mathbf{X}(u, v)$ and let $\mathbf{v}$ and $\mathbf{w}$ be two vectors in $T(P)$,

$$I(\mathbf{v}, \mathbf{w}) = \mathbf{v}^T \begin{bmatrix} E & F \\ F & G \end{bmatrix} \mathbf{w},$$

with $E =< \mathbf{X}_u, \mathbf{X}_u >$, $F =< \mathbf{X}_u, \mathbf{X}_v >$ and $G =< \mathbf{X}_v, \mathbf{X}_v >$, where $\mathbf{X}_u$ and $\mathbf{X}_v$ denote the first derivatives of the parametric function $\mathbf{X}$ computed at $(u, v)$ and $< \cdot, \cdot >$ denotes the inner product in $\mathbb{R}^3$. The first derivatives of $\mathbf{X}$ at $(u, v)$ span the tangent space $T(P)$, so the surface normal at $P$ can be defined in terms of their cross product:

$$\mathbf{n} = N_{\mathcal{S}}(P) = \frac{\mathbf{X}_u(u, v) \times \mathbf{X}_v(u, v)}{|\mathbf{X}_u(u, v) \times \mathbf{X}_v(u, v)|}.$$

The *second fundamental form* is a tensor defined by projecting the second partial derivatives of $\mathbf{X}$ on the normal direction $\mathbf{n}$, which is represented by the matrix:

$$II = \begin{bmatrix} L & M \\ M & N \end{bmatrix}$$

where $L =< \mathbf{X}_{uu}, \mathbf{n} >$, $M =< \mathbf{X}_{uv}, \mathbf{n} >$, $N =< \mathbf{X}_{vv}, \mathbf{n} >$ and $\mathbf{X}_{uu}$, $\mathbf{X}_{uv}$, and $\mathbf{X}_{vv}$ denote the second partial derivatives of $\mathbf{X}$ computed at $(u, v)$.

The partial derivatives of $\mathbf{n}$ with respect to $u$ and $v$ are defined in terms of the coefficients of the first and second fundamental forms by the Weingarten equations

$$\mathbf{n}_u = \frac{FM - GL}{EG - F^2}\mathbf{X}_u + \frac{FL - EM}{EG - F^2}\mathbf{X}_v$$

$$\mathbf{n}_u = \frac{FN - GM}{EG - F^2}\mathbf{X}_u + \frac{FM - EN}{EG - F^2}\mathbf{X}_v$$

from which we get the following expression of the shape operator at $P$:

$$S_P = (EG - F^2)^{-1} \begin{pmatrix} LG - MF & ME - LF \\ ME - LF & NE - MF \end{pmatrix}.$$

On a sufficiently small neighborhood of $P$, the surface can be expressed in terms of an explicit function defined on a frame $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ with origin at $P$ and $\mathbf{w}$ axis not parallel to the tangent plane $T(P)$. Note that $\mathbf{w}$ needs not be aligned with the surface normal $\mathbf{n}$. In this case, the parametric function $\mathbf{X}$ has the form

$$\mathbf{X}(u, v) = (u, v, f(u, v)),$$

where $f(u, v)$ is a bivariate scalar function. The coefficients of the first and the second fundamental form, hence the shape operator, are easily derived from the first and second partial derivatives of $f$.

A point $P$ is said to be *regular* if the two principal curvatures at $P$ are different. Each principal curvature forms a *line field* that is defined at all regular points and rules the surface. The points where the two principal curvatures are equal correspond to singularities of the curvature line fields and they are called *umbilical points*. Principal directions are undefined at umbilical points.

Consider the principal curvatures $k_1$ and $k_2$ as scalar fields defined on $\mathcal{S}$. Then, the gradient $\nabla k_i$ of curvature $k_i$ is a vector field on the tangent bundle of $\mathcal{S}$. The *extremality* $e_i$ at a regular point $P$ is defined as the inner product between the gradient of curvature $k_i$ and its related direction, i.e.,

$$e_i =< \nabla k_i, \mathbf{t_i} >= \frac{\partial k_i}{\partial \mathbf{t_i}} \tag{1}$$

where all quantities are evaluated at $P$. Note that the sign of $e_i$ is not well defined, since it depends on an arbitrary orientation of $\mathbf{t}_i$. *Creases* are defined as those regular points where extremalities vanish, with the following additional constraints:

$$e_1 = 0 \quad \wedge \quad \frac{\partial e_1}{\partial \mathbf{t_1}} < 0 \quad \wedge \quad k_1 > |k_2| \tag{2}$$

$$e_2 = 0 \quad \wedge \quad \frac{\partial e_2}{\partial \mathbf{t_2}} > 0 \quad \wedge \quad k_2 > -|k_1|. \tag{3}$$

Creases defined by $e_1$ are also called *ridges*, while creases defined by $e_2$ are also called *valleys.*

## 4 Curvature estimation

We take in input a mesh of triangles $M$ and we evaluate the shape operator, hence the principal curvatures and curvature directions, at each vertex $P$ of $M$ by using a surface fitting method, which is a variation of the *Osculating Jets* proposed in [Cazals and Pouget 2005a; Cazals and Pouget 2008].

The original method of [Cazals and Pouget 2005a] can be briefly summarized as follows:

1. gather the vertices of $M$ in a neighborhood of $P$ (usually just a few rings around $P$, depending on the selected degree of the polynomial to be fitted);
2. set a local frame $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ - hereafter called the *fitting frame* - centered at $P$ with its $\mathbf{w}$ axis not parallel to the tangent plane at $P$ (an early implementation used a coordinate axis, while a later implementation selects a better fitting frame by performing principal component analysis (PCA) of the neighborhood);
3. express gathered data in the local frame and fit a polynomial $f(u, v)$ of given degree $m$, by resolving a least square problem (usually with Singular Value Decomposition);

4. evaluate the shape operator, hence the principal curvatures and directions, at $(0, 0, f(0, 0))$ in the local frame and set those values to estimate the curvatures at $P$.

In our approach, we wish to estimate the local shape at various scales by fitting a polynomial on a more or less extended neighborhood of $P$. Similarly to [Yang et al. 2006], we use the radius $r$ of a neighborhood as a scale parameter. A fundamental assumption of the Osculating Jets is that the input surface can be expressed in explicit form in the neighborhood $B$ of a point $P$. If this assumption is violated, then the method may become highly inaccurate. This hardly happens in small rings, but it is likely to occur, even with slightly large neighborhoods, in the proximity of regions with a high curvature. Since we wish to evaluate the curvature at quite large scales, i.e., in the order of a few tenths of the diameter of the bounding box of an object, we cannot be careless in gathering data from a neighborhood. Thus, given a vertex $P$ and a scale parameter $r$, we proceed as follows:

- We perform a breadth-first traversal of the mesh, starting at $P$, until we get vertexes that lie in the ball of radius $r$ centered at $P$, and we gather all such vertices in a set $V_P$;
- Next we set the $\mathbf{w}$ axis of the fitting frame to be equal to the average $\frac{1}{|V_P|}\sum_i \mathbf{n}_i$, where summation is on all elements of $V_P$ and $\mathbf{n}_i$ is a pseudo-normal of the surface estimated at $v_i$ with a standard method (e.g., a weighted average of surface normals of triangles in its 1-ring). Pseudo-normals are estimated once and for all during pre-processing.
- For all $i$, we compute the scalar product $< \mathbf{n}_i, \mathbf{w} >$ and we discard from $V_P$ each vertex $v_i$ giving a negative value (i.e., vertices corresponding to a flipping portion of surface).

Note that pseudo-normals $\mathbf{n}_i$ are *not* used to set the fitting frame at their respective vertexes. They are just used to either accept or discard vertexes in the neighborhood $B$. Now we are left with a relatively large set of vertexes to be plugged into a least square problem. We improve efficiency and scalability by acting on the degree of the polynomial and on the size of the data set.

Since we are wish to extract creases, a logical choice would be to fit polynomials of degree at least three (for instance, polynomials of order four are used in [Cazals and Pouget 2008]). However, polynomials of degree three [four] would give us a least square problem with ten [fifteen] unknowns. We can speed-up computation an order of magnitude by fitting a polynomial of degree two, which is sufficient to evaluate second order differential quantities, while relying on a discrete approach for extracting higher order differential quantities. Moreover, we constrain such a polynomial to go through point $P$, thus forcing the coefficient of order zero to vanish. This reduces the unknowns to just five coefficients, i.e.,

$$f(u, v) = au^2 + bv^2 + cuv + du + ev. \qquad (4)$$

The loss of accuracy with respect to fitting a complete polynomial is compensated by the fact that our polynomial goes exactly through $P$, thus it is not necessary to approximate the differential quantities at $P$ with those computed at its footprint on the graph of $f$.

Second, we perform Monte-Carlo sampling on the set of data $V_P$. In fact, the amount of data necessary to obtain e

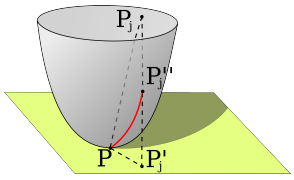



**Figure 2:** *The quadric surface approximating the (smoothed) surface in the neighborhood of a vertex $P$, obtained during curvature estimation. The arc length of the curve joining $P$ to $P_j''$ is used to compute an approximation of the true distance between $P$ and $P_j$ on the smoothed surface.*

reliable fit is not a function of the size of the neighborhood, but rather a function of the number of unknowns, which is fixed to five, provided that data are sampled uniformly in the neighborhood. Therefore, we can afford sampling a relatively small set of points. In Section 6, we analyze the variation in estimates of curvatures and curvature directions as a function of the number of sampled points, and we show that sampling about 50 vertexes gives excellent results.

With these modifications, the Osculating Jets becomes robust and fast enough to be used for multi-scale curvature estimation even at large scales and with very noisy data, as we show in Section 6.

## 5 Extraction of creases

Once principal curvatures and curvature directions have been extracted, information we need for extracting creases already come at the proper scale. This means that we do not need to extend our computations to a large neighborhood. However, we must be careful to make small use of mesh geometry, because this has *not* been smoothed, while it still refers to the finest scale. Therefore, we develop a discrete method that, at each vertex, makes use only of information from its 1-ring, and relies on geometry of the quadric surfaces fit to data during curvature computation, rather than on the original mesh.

Let $k_i : \mathcal{S} \to \mathbb{R}$, for $i = 1, 2$ be the fields of principal curvature, estimated at all vertexes of mesh $M$. We first estimate their gradients $\nabla k_i : \mathcal{S} \to T_{\mathcal{S}}$, where $T_{\mathcal{S}}$ denotes the tangent bundle of $\mathcal{S}$.

Let $P$ be a vertex of $M$ and let $P_j$, for $j = 1 \ldots h$, be its neighbors. Let $P_j'$ be the projection of $P_j$ on the tangent plane $T(P)$ and let $\mathbf{t}_j$ be the direction of $P_j' - P$. Note that this is the only datum from the geometry of $M$ that we use in our computation. Since mesh smoothing displaces vertexes essentially in the normal direction, this projection is not likely to change much through the scales. We estimate the derivative of $k_i$ along $\mathbf{t}_j$ with a finite difference, thus obtaining the following equation:

$$< \nabla k_i(P), \mathbf{t}_j >= \frac{k_i(P_j) - k_i(P)}{d(P, P_j)}, \qquad (5)$$

where $d(P, P_j)$ denotes a distance between $P$ and $P_j$. We collect such equations for all $j = 1 \ldots h$ and we obtain the components of $\nabla k_i$ by resolving the corresponding least square problem with two unknowns and $h$ equations.

Note that distance $d(P, P_j)$ in Equation 5 should be measured on the (unknown) smoothed version of $\mathcal{S}$ at scale $r$. The best approximations that we have of that surface in the

proximity of $P$ and $P_j$ are provided by the quadric functions that we used at $P$ and $P_j$, respectively, during curvature estimation, i.e.,

$$w = \frac{k_1}{2}u^2 + \frac{k_2}{2}v^2 \quad (6)$$

where the equation is expressed in the Monge frame at either $P$ or $P_j$, respectively, and the values of $k_1$ and $k_2$ are taken at the corresponding point. Given one of these two surfaces - say the one centered at $P$ - we measure the arc length on this quadric surface between $P$ and the vertical projection of $P_j$ on the same surface (see Figure 2). This value is computed analytically by resolving a line integral on the conic line obtained by sectioning the surface with the vertical plane through direction $\mathbf{t}_j$. The formula can be derived easily through a solver, like *Maxima*, and it involves Equation 6 as well as the coordinates of $P'_j$. We do not report it here for brevity. We repeat the same computation by considering the surface centered at $P_j$ and we compute the average between the two arc lengths.

Once the gradients of the principal curvatures have been computed at each vertex, our method proceeds similarly to that of [Hildebrandt et al. 2005]:

1. Compute extremalities through Equation 1;
2. Extract creases triangle by triangle, by setting the orientation of principal axes at the vertexes, so that corresponding axes form acute angles (see also [Cazals and Pouget 2005b] about the "acute rule") and recomputing the signs of extremalities accordingly. The sign of the derivative of extremalities, which is required in Equations 2 and 3, is computed by finite differences along two edges of $t$, by applying the same method as in Equation 5, where $k_i$ is replaced with $e_i$. Each triangle $t$ can contain at most one crease segment, whose endpoints are computed by linear interpolation on the edges of $t$;
3. Trace creases to form polylines.

Two optional steps can be performed to improve the shape of creases: Laplacian smoothing on the field of extremalities can be performed before Step 2; and creases can be filtered after Step 3, as suggested in [Ohtake et al. 2004]. Creases are filtered by compting the line integral of curvature magnitudes along each polyline, and discarding polylines with a value below a given threshold.

## 6 Experimental results

We implemented our methods in C++ by using the VCG Library [VCG ] for geometric data structures and the Eigen library [Eig ] for numerical computations. Experiments were run on a PC with a 2.67Ghz Core i5 processor equipped with 4Gb of memory, using a single core. We tested our algorithms against other methods at the state-of-the-art on some of the data sets available in the public domain for benchmarks.

### 6.1 Curvature

We first consider smooth datasets to extract curvatures at various scales. In each mesh, we use the average length of edges $e$ as a reference to set the scale $r$ for fitting. An example of curvature extracted at various scales from a large mesh is reported in Figure 3. The color map combines principal curvatures with a non-linear mapping, designed to enhance

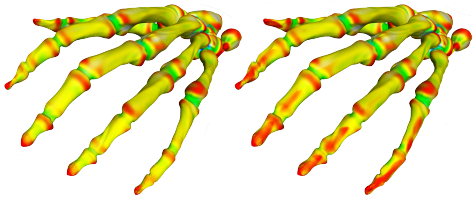


**Figure 4:** *Principal curvatures computed with our method (left) and with Osculating Jets (right). Scale is* $16e$*, which is about twice the diameter of the fingers. The Osculating Jets incorrectly classifies as convex (red) some cylindrical parts (yellow).*

variations also at small curvatures. Color codes are as follows: red convex; yellow flat-convex; green saddle; cyan flat-concave; blue concave; white flat. At the finest scale the curvature map enhances the artifacts of object reconstruction on the blade, as well as the fine detail of the rugged bottom part of the object. Curvature of such details, as well as small details on the edges of the blade, and bas-relief letters on the bottom part, progressively disappear at the larger scales, while the curvature of larger details is correctly characterized throughout all scales.

We compare our method for curvature extraction with the classical Osculating Jets and with the integral invariant method of Pottmann et al. Curvatures are estimated at various scales, ranging from twice to 16 times the average edge length in the mesh. Our method is applied by performing Monte-Carlo sampling with a threshold of 50 vertexes.

An implementation of the classical Osculating Jets with degree two is derived from the implementation of our method, the most important difference being that the whole neighborhood is always used for fitting. We do not report running times for this method, because they are quite similar to ours.

For the integral invariant method, we use the implementation provided by the authors, which is based on efficient computation of PCA on ball neighborhoods via FFT. The method performs a space discretization, which has high memory requirements. The program is an executable for Windows that cannot work beyond the allowed threshold of 2Gb of memory. Therefore, we could not run it on any dataset with a discretization step smaller than 0.005 times the size of the bounding box. This fact puts a severe lower bound on the meaningful scales that can be used: we have run experiments with this method only when the scale was not finer than 5 times the size of the voxel, i.e., 0.025 times the size of the bounding box.

Numerical results are reported in Table 1, while visual results for some examples are shown in Figure 4 and 5.

The standard Osculating jets behaves similarly to our algorithm at small enough scales, but it produces artifacts near small details at higher scales, where a large enough neighborhood captures also portions of surface that are flipped with respect to the fitting plane. Some such artifacts can be seen in Figure 4, where Osculating Jets marks as convex large cylindric parts of the mesh near the fingertips.

As shown in Figure 5, the discretization scale of the integral invariant method is too coarse to produce meaningful results at a fine scale $2e$, while our method correctly detects the curvature of small features (note for example the bas-relief letters). At scale $4e$ discretization artifacts still appear (diagonal stripes on the flat part of the object). At larger scales,

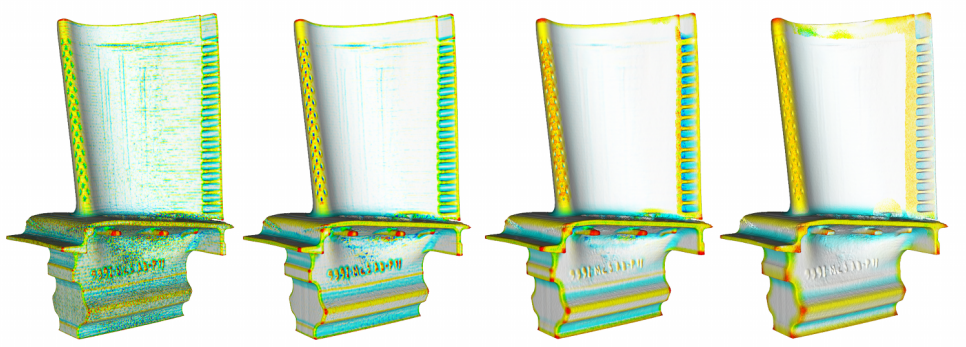


**Figure 3:** *Principal curvatures computed on the turbin blade dataset at scales* $2e$*,* $4e$*,* $8e$ *and* $16e$*.*

| Dataset | $\|M\|$ | 2e | | 4e | | 8e | | 16e | |
|---|---|---|---|---|---|---|---|---|---|
| Gargoyle | 25k | 0.4 | - | 1.1 | 12.9 | 4.7 | 13.1 | 22.9 | 13.7 |
| RockerArm | 35k | 0.5 | - | 1.8 | 14.0 | 3.8 | 14.1 | 28.1 | 14.6 |
| David head | 50k | 0.9 | - | 2.6 | 30.7 | 9.7 | 31.7 | 37.6 | 30.5 |
| Angel | 237k | 4.7 | - | 13.1 | - | 50.0 | - | 220.2 | - |
| Turbin Blade | 883k | 12.9 | - | 33.7 | - | 128.0 | 22.9 | 809.6 | 23.5 |

**Table 1:** *Running times (in seconds) for extracting curvatures on various datasets at different scales with our method (left columns) and with the integral invariant method (right columns). Sizes of the datasets are given by the number of vertexes (in thousands). Scales are expressed in terms of the average edge length* $e$*. The Angel dataset could not be loaded with the integral invariant program.*

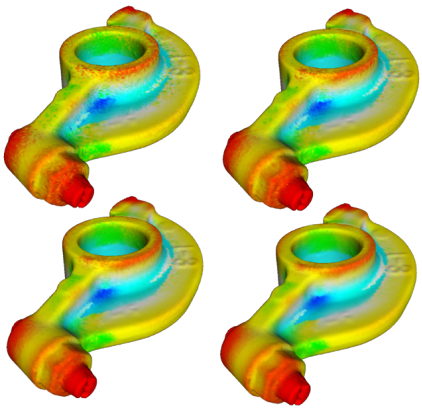


**Figure 6:** *Curvature extracted at scale* $16e$*. Top: Monte-Carlo with 50 samples (left) and 100 samples (right); bottom: Monte-Carlo with 200 samples (left) and full neighborhood using about 1200 data points on average (right).*

both methods produce very smooth results, which are comparable and compatible with object features at that scale. In terms of running times, the integral invariant method is competitive on large objects at large scales, while our method is faster at small scales and for objects of moderate size.

Next we show the effect of applying our method with Monte-Carlo sampling against using the whole neighborhood. Since we do not have any ground truth for curvatures on these data sets, we take the computation with the whole neighborhood as a reference, and we report the deviation from those results at different rates of Monte-Carlo sampling. Numerical results are reported in Table 2, while visual results are shown in Figure 6. Note that the loss of accuracy is quite small, even at large scales, by using 100 samples, while results with 200 samples are almost identical to those obtained with the whole neighborhood, which fits to about 1200 data points on average.

As expected, Monte-Carlo sampling drastically reduces the time required for fitting at large scale. Unfortunately, computation times are dominated from the breadth-first traversal of the mesh, hence the global speed-up is just moderate. We believe that a careful implementation of the search with ad hoc data structures could drastically reduce computation times, thus achieving a better scalability.

Next we test robustness by adding noise in the normal direction to vertexes. Noise is generated by extracting random values in a range $[-\sigma, \sigma]$, where $\sigma$ is defined as a fraction of the average edge length in the mesh. We show results with $\sigma = 0.25e, 0.5e, 1.0e$ at scales $8e$ and $16e$. Our method is compared just to the integral invariant method in this case. Each method takes as reference values the estimates it obtains on the noiseless version of each dataset at the same scale. Numerical results, reported in Table 3, show that the two methods produce similar results, the integral invariant performing slightly better for large error.

### 6.2 Creases

Finally, we test the performance of our method for extracting creases against (our implementation of) the method proposed in [Hildebrandt et al. 2005]. While the two methods produce comparable results on clean data and at small scales, our method performs much better than the other one when data are noisy, or scale is large, or both. Creases extracted with the method of [Hildebrandt et al. 2005] from noisy data may be highly fragmented and they are generally more irregular. Moreover, they do not always merge correctly to follow large scale features. A visual comparison at scale $8e$is reported in Figure 7. In the clean dataset (top row), note how our method correctly merges the parallel ridges around the rim of the hole, and along the left rib, while such ridges remain separated with the other method. Note that creases are drawn on the original mesh, thus they necessarily follow the jagged surface in the noisy case.

An example of the behavior of creases through different scales is reported in Figure 8. Note how creases slide through the surface, merge and disappear by increasing scale. For instance: the nose ridge is marked by two creases

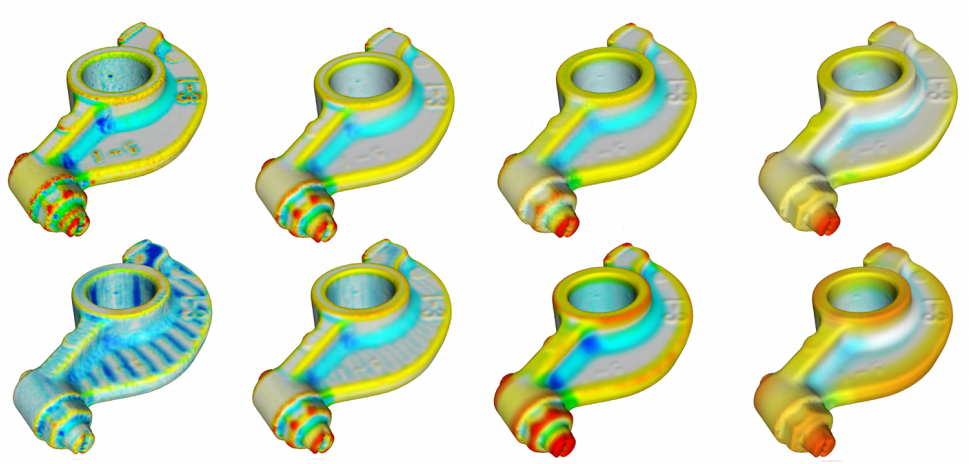


**Figure 5:** *Curvature extracted from the rockerarm dataset at scales* $2e$, $4e$, $8e$, $16e$. *Top row: our method; bottom row: integral invariant method.*

| | | Time w/o M-C | | Number of Monte-Carlo samples | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 50 | | | | 100 | | | | 200 | | | |
| Dataset | $\|M\|$ | Fit | Total | $\bar{x}$ | $\sigma$ | Fit | Total | $\bar{x}$ | $\sigma$ | Fit | Total | $\bar{x}$ | $\sigma$ | Fit | Total |
| Gargoyle | 25k | 3.4 | 23.0 | 0.95 | 0.13 | 0.2 | 20.0 | 0.97 | 0.09 | 0.3 | 20.3 | 0.98 | 0.06 | 0.5 | 20.5 |
| RockerArm | 35k | 3.7 | 28.3 | 0.97 | 0.09 | 0.3 | 25.2 | 0.99 | 0.06 | 0.4 | 25.4 | 0.99 | 0.04 | 0.7 | 25.7 |
| David head | 50k | 7.1 | 37.3 | 0.96 | 0.12 | 0.4 | 31.3 | 0.98 | 0.08 | 0.6 | 31.6 | 0.99 | 0.06 | 1.0 | 32.0 |
| Angel | 237k | 29.5 | 220.0 | 0.98 | 0.08 | 1.9 | 195.3 | 0.99 | 0.06 | 2.9 | 196.7 | 0.99 | 0.04 | 4.9 | 198.9 |
| Turbin Blade | 883k | 108.6 | 809.5 | 0.96 | 0.12 | 6.7 | 719.0 | 0.98 | 0.09 | 10.4 | 724.3 | 0.99 | 0.07 | 18.0 | 733.1 |

**Table 2:** *Average deviation, variance and computational time using different numbers of Monte-Carlo samples at scale* $32e$. *Deviation is measured as the scalar product between the principal directions of curvature: 1.0 means perfect match. Time is in seconds.*

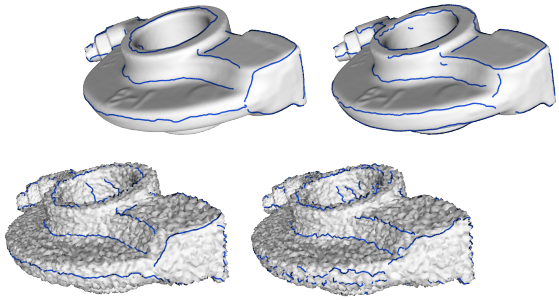


**Figure 7:** *Creases computed at scale* $8e$ *with our method (left) and with the method of [Hildebrandt et al. 2005] (right). Clean data (top) and noisy data with* $\sigma = 1.0e$ *(bottom).*

at fine scale, which soon merge into one crease that persists throughout the following scales. At the highest scale, this crease extends to mark the bilateral symmetry of the object. Creases that outline the arms of the Moai statue are detected at the fine scale, then they disappear. At fine scales, the body of the statue is outlined longitudinally by several creases, which progressively merge, ending up to the central longitudinal crease. Transverse creases at the eyebrows and at the top of the forehead first merge, then disappear when scale gets bigger; the same happens with other transverse creases, such as those marking the chin, the nostrils, and below the nose.

## 7 Concluding remarks

We have shown that fitting methods can be accurate in curvature estimation at different scales, provided that the fitting frame is chosen carefully and portions of surface bending away are discarded. We have also shown that Monte-Carlo sampling can reduce the time needed for fitting, without much loss of accuracy. However, at large scales, computation is still dominated by the time needed to extract vertex neighborhoods. This problem can be probably resolved by developing suitable data structures for mesh traversal. We will address this specific problem in our future work.

We have presented a new discrete method for extracting creases that is accurate and robust, and combines nicely with our multi-scale curvature estimation. We have demonstrated on some examples how creases subsume shape features at the various scales: they may either disappear, or slide over the surface as scale increases, and they may eventually merge.

It would be interesting to compare our results with respect to creases extracted from progressively smoothed versions of the same surface, e.g., with Gaussian smoothing applied at different scales. More generally, it is an open issue to establish a formal mathematical relation between the radius $r$ of the neighborhood used to evaluate curvature and the scale of a corresponding smoothing filter.

Our goal in the near future is to build a scale-space for surface features, which can provide a flexible tool for shape analysis and processing. In the scale space, creases are characterized with both position and persistence through scales. Since creases evolve in a quite complicated way, tracing them through the various scales is likely to be a challenging task.

## Acknowledgements

We wish to thank the authors of [Yang et al. 2006] for providing the software to compute multi-scale curvatures with

| $\sigma$ | 0.25e | | | | 0.50e | | | | 1e | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scales | 8e | | 16e | | 8e | | 16e | | 8e | | 16e | |
| RockerArm | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.96 | 0.99 | 0.97 | 0.98 | 0.81 | 0.99 |
| David head | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.99 | 0.99 | 0.99 | 0.93 | 0.97 | 0.94 | 0.98 |

**Table 3:** *Average deviation of curvature directions at different scales and different noise $\sigma$. The error made by our method (left columns), and by the integral invariant method (right columns) is measured as the scalar product between the principal directions of curvature.*

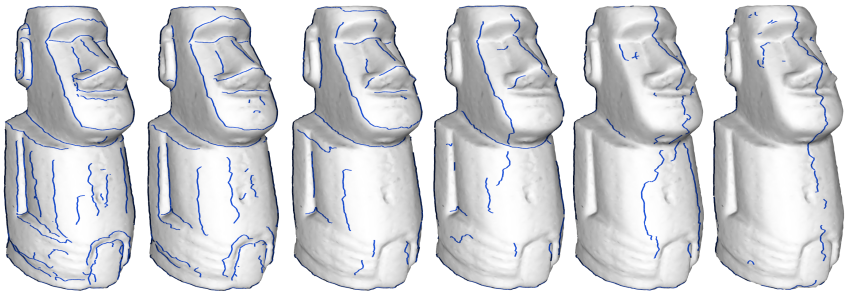


**Figure 8:** *Crease slide through the surface, merge and disappear as the scale increases.*

the integral invariant method.

## References


AGAM, G., AND TANG, X. 2005. A sampling framework for accurate curvature estimation in discrete surfaces. *IEEE Transactions on Visualization and Computer Graphics 11*, 5, 573–583.

CAZALS, F., AND POUGET, M. 2005. Estimating Differential Quantities using Polynomial fitting of Osculating Jets. *Computer Aided Geometric Design 22*, 121–146.

CAZALS, F., AND POUGET, M. 2005. Topology driven algorithms for ridge exctraction on meshes. Tech. Rep. 5526, INRIA.

CAZALS, F., AND POUGET, M. 2008. Algorithm 889: Jet_fitting_3: A generic c++ package for estimating the differential properties on sampled surfaces via polynomial fitting. *ACM Trans. Math. Softw. 35*, 3, 1–20.

COHEN-STEINER, D., AND MORVAN, J.-M. 2003. Restricted delaunay triangulations and normal cycle. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, ACM, New York, NY, USA, 312–321.

COSTA BATAGELO, H., AND WU, S.-T. 2007. Estimating curvatures and their derivatives on meshes of arbitrary topology from sampling directions. *The Visual Computer 23*, 9, 803–812.

DESBRUN, M., GRINSPUN, E., AND SCHRÖDER, 2005. Discrete differential geometry: an applied introduction. ACM SIGGRAPH 2005 Course Notes.

DOUROS, I., AND BUXTON, B. 2002. Three-dimensional surface curvature estimation using qudric surface patches. In *Proceedings Scanning 2002*.

Eigen. `http://eigen.tuxfamily.org/`.

GATZKE, T. D., AND GRIMM, C. M. 2006. Estimating curvature on triangular meshes. *International Journal on shape Modeling 12*, 1–29.

GOLDFEATHER, J., AND INTERRANTE, V. 2004. A novel cubic-order algorithm for approximating principal direction vectors. *ACM Trans. Graph. 23*, 1, 45–63.

GRINSPUN, E., GINGOLD, Y., REISMAN, J., AND ZORIN, D. 2006. Computing discrete shape operators on general meshes. *Computer Graphics Forum 25*, 547–556.

HILDEBRANDT, K., POLTHIER, K., AND WARDETZKY, M. 2005. Smooth feature lines on surface meshes. In *Proc. 3rd Eurographics Symp. on Geom. Proc.*, 85.

KOENDERINK, J. 1994. *Scale-space theory in computer vision*. Kluwer Academic.

LINDBERG, T. 2009. Scale-space. In *Encyclopedia of Computer Science ad Engineering*, B. Wah, Ed. John Wiley and Sons, 2495–2504.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, 609–612.

PAULY, M., KOBBELT, L. P., AND GROSS, M. 2006. Point-based multiscale surface representation. *ACM Trans. Graph. 25*, 2, 177–193.

PORTEOUS, I. 2001. *Geometric Differentiation (2nd edition)*. Cambridge University Press.

POTTMANN, H., WALLNER, J., YANG, Y.-L., LAI, Y.-K., AND HU, S.-M. 2007. Principal curvatures from the integral invariant viewpoint. *Comput. Aided Geom. Des. 24*, 8-9, 428–442.

Vcglib. `http://vcg.sourceforge.net/`.

YANG, Y.-L., LAI, Y.-K., HU, S.-M., AND POTTMANN, H. 2006. Robust principal curvatures on multiple scales. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 223–226.

YOSHIZAWA, S., BELYAEV, A., AND SEIDEL, H.-P. 2005. Fast and robust detection of crest lines on meshes. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, ACM, New York, NY, USA, 227–232.

# Advances in Metric-neutral Visualization

Charles Gunn
Institüt der Mathematik MA 3-2
Technische Universität Berlin

**ABSTRACT**

We describe a visualization system in which the two classical noneuclidean spaces – elliptic and hyperbolic – are integrated as equal citizens along with euclidean space. Such software we call metric-neutral. After surveying previous work in this direction, we review the mathematical foundations, particularly the projective models for these spaces. We give an overview of the issues involved in converting euclidean visualization software to be metric-neutral, beginning with non-interactive issues before turning to interaction, and finally, to immersive environments. We describe how the metric-neutral visualization system under discussion solves these challenges, highlighting a number of innovative features, including metric-neutral tubing, metric-neutral realtime shading, and metric-neutral tracking.

**Keywords:** projective geometry, noneuclidean geometry, noneuclidean tracking, curved spaces, metric-neutral software, visualization, Cayley-Klein geometry

## 1 INTRODUCTION

The discovery of noneuclidean geometries in the nineteenth century is one of the most exciting and important chapters in modern mathematics. It has had significant consequences in the development not only of mathematics itself but also natural science and philosophy. The alternative experience of space provided by these geometries exerts a fascination accessible to non-mathematicians. The circle-limit prints of the M. C. Escher have helped popularize the underlying concepts. There is a widening circle of scientific research based on noneuclidean geometry, ranging from cosmology ([Wee90]) to the study of large graphs ([Mun98]) to the classification of 3D manifolds ([Thu97]) to the perceived structure of the human visual experience ([Hee83]).

The current work is the outgrowth of research centered on the challenge of visualizing three-dimensional manifolds and orbifolds with geometric structures ([Gun93]). The recent solution of the Poincare Conjecture and the more general Geometrization Conjecture ([Mac06]) establishes that all three-dimensional manifolds can be decomposed into submanifolds that have geometric structures. Hence, software systems such as the one described in this article can be used to visualize *all* three-dimensional manifolds.

In this article we present a unified approach to visualization of these geometries alongside euclidean geometry, based on their common ancestry within projective geometry. We show how through this approach, much of the theory and practice of euclidean visualization science can be transferred with minimal effort to these noneuclidean spaces. We first survey previous work in this direction, and then give a review of the essential mathematics which underlies the current work.



### 1.1 Comparison with previous work

The current work builds upon the theory and practice described in [Gun92], [PG92], [Gun93] and [Wee02]. This article goes beyond existing literature by introducing the concept of *metric neutrality*. Our discussion of metric neutrality provides software practitioners for the first time a theoretical and practical framework for upgrading a general-purpose euclidean visualization system to handle noneuclidean geometries as equal citizens.

The visualization system used to implement the metric-neutral ideas presented in this article is jReality ([jr06], [WGH$^+$09]), an open-source, Java-based, general-purpose 3D scene graph package. Geomview ([MLP$^+$]) was an earlier attempt in the direction of metric-neutral visualization system dating from the early 1990's. jReality extends Geomview in a number of ways, which are described in the course of the article.

Earlier attempts to visualize noneuclidean spaces in immersive environments ([GH97], [FGK$^+$03]), retained standard euclidean tracking. The present work describes and implements a noneuclidean tracking solution which significantly improves the quality of the noneuclidean immersive experience.

This article only considers metrics of constant curvature. The extension of this approach to non-constant curvature manifold visualization ([WSE04]) is a natural goal for further work.

## 2 NONEUCLIDEAN GEOMETRY

One of the standard practices of computer graphics is the use of homogeneous coordinates to represent points

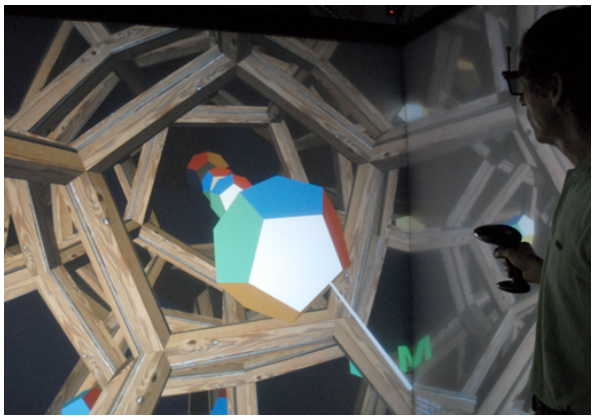


Figure 1: Immersive experience of elliptic space.

in euclidean space. A point $P = (x,y,z) \in \mathbf{R}^3$ is assigned the homogeneous coordinates $(x,y,z,1)$. The term *homogeneous* comes from the equivalence relation given by $(x,y,z,1) \cong (\lambda x, \lambda y, \lambda z, \lambda)$ for $\lambda \in \mathbf{R}, \lambda \neq 0$. Computer graphics practicioners learn that, using homogeneous coordinates, every euclidean isometry can be represented as a single $4x4$ matrix.

Homogeneous coordinates also play a crucial role in the perspective transformation of computer graphics. In the typical case, the camera position $(0,0,0,1)$ is mapped to the point $(0,0,1,0)$. The latter point is not equivalent to any point in $\mathbf{R}^3$! The practical result is that the trapezoidal viewing frustum is mapped to the familiar rectangular box form for 3D normalized device coordinates from which a rendered image can conveniently be calculated.

A search for the deeper significance of homogeneous coordinates leads to an important chapter in the history of mathematics. Homogeneous coordinates are the natural coordinates for *projective geometry*, a branch of mathematics developed in response to the birth of perspective painting. Projective geometry includes a set of points such as the point $(0,0,1,0)$ mentioned above which are *not* elements of $\mathbf{R}^3$, the so-called ideal points or points at infinity. The inclusion of these new points results in a geometry in which euclidean measurement is no longer possible. Analogous to a motion of euclidean space which preserves distances, a *projectivity* is a transformation of projective space which maps lines to lines and preserves geometric incidence.

At the same time as projective geometry was developed in its modern form, independent research established the existence of other metric geometries. Euclidean geometry is distinguished by one of its axioms, the so-called Parallel Postulate. A logically equivalent form states: given a point and a line in a plane, there is exactly one line in the plane passing through the point which is parallel to the given line.The discovery of other geometries was based on the demonstration that this postulate can be replaced by two different alternatives, and the resulting *geometry* is a consistent system, satisfying the other axioms. The two alternatives are first, that there are no such parallels (yielding elliptic geometry) or that there are infinitely many (yielding hyperbolic geometry). The credit for this discovery was shared by Gauss, Bolyai, and Lobachevsky.

Projective geometry was originally developed synthetically (without coordinates). It was then shown that one could begin with homogeneous coordinates and arrive at projective geometry. This geometry exists in every dimension $n > 0$ and is written $\mathbf{R}P^n$. Closely related to $\mathbf{R}P^n$ is $PGL(n+1,\mathbf{R})$, the group of all invertible $(n+1)$ by $(n+1)$ matrices, subject to a homogeneous equivalence relation (hence the $P$ in the name). The elements of this matrix group act on points of $\mathbf{R}P^n$ by matrix multiplication.[1] Every projectivity can be represented by an element of $PGL(n+1,\mathbf{R})$, and vice-versa.

The final important step in the history came as Cayley discovered, in his words, "Projective geometry is all geometry." He did this – in modern terminology and restricting to the case $n = 3$ – by introducing a quadric surface, termed the Absolute, within $\mathbf{R}P^3$. He showed different choices for the Absolute lead to models for euclidean, elliptic, and hyperbolic geometry *within* projective geometry. The mathematical details related to this construction have been collected in Appendix A.

## 2.1 Isometry groups

All projectivities which preserve the metric relationships established by the Absolute, form a group called the *isometry* group of the geometry. The isometry groups of these three geometries are then subgroups of $PGL(n+1,\mathbf{R})$. The isometry group for elliptic space is $SO(4)$ while that of hyperbolic space is $SO(3,1)$ familiar as the isometry group of Minkowski space in relativity theory.[2] The euclidean group $SE(3)$ is more complicated than the two noneuclidean cases, since the euclidean metric involves a degenerate quadric and requires a delicate limiting process to be fully defined.

All three groups are 6-dimensional Lie groups which contain the rotation group $SO(3)$, fixing the point $(0,0,0,1)$, as a subgroup. Here's a few facts about non-euclidean isometries needed for later. A non-euclidean isometry[3] is characterized by two invariant lines, the *axes* of the isometry, which are polar pairs (see Appendix A) with respect to the metric quadric. The isometry can be factored as the product of two (commuting) rotations around these axes. This is in general a screw motion. A *rotation* around a line $l$ is

[1] Since the points are homogeneous, it is immediately obvious that the matrices which act on the points also can be multiplied by an arbitrary non-zero factor without disturbing the equivalence relation defined on the points.

[2] Note Minkowski space does not use homogeneous coordinates hence is a true 4-dimensional space.

[3] with the exception of so-called Clifford translations in elliptic space

then an isometry in which $l$ and its polar line $\hat{l}$ are the axes, such that the rotation around $\hat{l}$ is the identity. A *translation* carrying a point $A$ to another point $B$, on the other hand, is an isometry with the lines $l := \overleftrightarrow{AB}$ and its polar line $\hat{l}$ as axes, such that the rotation around $l$ is the identity.

## 2.2 Elliptic space and spherical space

In popular accounts of noneuclidean geometry, elliptic space $\mathbf{Ell}^n$ and spherical space $\mathbf{S}^n$ are sometimes not clearly distinguished. Mathematically, $\mathbf{S}^n$ is the simply-connected covering space of $\mathbf{Ell}^n$, and can be decomposed as two copies of $\mathbf{Ell}^n$. For $n = 2$, for example, the ordinary sphere can be decomposed as two hemispheres (each one a $\mathbf{Ell}^2$). Spherical space does not satisfy the axiom (inherited from Euclid) that two lines intersect in at most one point (the lines of spherical space are great circles which always intersect in *two* antipodal points). For this reason, the existence of the sphere did not play a significant role in the discovery of noneuclidean geometry, even though with hindsight it can be used as an effective example. See Section 3.3 for visualization issues related to these two spaces.

# 3 NONEUCLIDEAN VISUALIZATION

Modern GPU's were designed to provide hardware support for perspective rendering of euclidean space. The euclidean subgroup and the perspective transformation, together, generate the full projective group $PGL(n+1,\mathbf{R})$. Hence, GPU's also can handle the isometries of the projective models of the noneuclidean geometries. What is required in order to make perspective renderings in these noneuclidean spaces as well? We first establish that the projective models of noneuclidean geometry have a special status in visualization science, based on their roots in perspective painting.

## 3.1 The insider's view

There are numerous models for noneuclidean geometry, for example, conformal models ([Thu97]). All models naturally each have their advantages and disadvantages. But in a certain sense – to be made more specific below – we argue that the projective one is the right one for a wide range of visualization tasks.

Visualization theory can be understood as an attempt to simulate images which an observer situated in the scene would actually see, or an embedded camera might form. We can think of such images as representing the *insider's* view of the scene – as opposed to images which might present another way of rendering the scene but not in a way in which such an imbedded observer would actually see via a perspective rendering. For example, in one dimension lower, standard images of a sphere imbedded in three-dimensional space do not represent the insider's view of the sphere since the camera lies *outside* the sphere itself.

Due to its close connection to perspective painting, the projective model described above is ideally suited to generate the insider's view. Consider first the perspective operation as a physical phenomena, for example, in a camera, in which paths of light (geodesics) through the center of perspective are mapped onto points of the image plane. Next, consider the perspective operation as it is implemented in hardware. The latter can be characterized as an operation in which *lines* through the center of perspective are transformed into points on the image plane.

The projective model is the only model in which these two conditions are naturally consistent. The other models all involve curved geodesics; any realistic rendering process must then first uncurve these geodesics before the hardware perspective operation can be applied. And even if one chooses to avoid the projective model altogether and ray trace with the curved geodesics, one arrives in the end at identical pictures to the ones which the projective model yields, since the insider's view is well-defined and independent of the model chosen to represent the geometry. Hence, the projective model and GPU technology are related as theory is to practice, and the resulting rendered images represent what an insider in these metric spaces sees. In the next section we demonstrate that this implies that cameras are projective, not metric, objects.

## 3.2 Cameras are projective, not metric

Once a camera is positioned within a scene and the scene has been shaded, the construction of a perspective image proceeds without any metrical considerations. True to its roots in perspective painting, the perspective transformation is a purely projective transformation, and can be applied as well in a noneuclidean space as in a euclidean space. Even the viewport, which we normally think of as a euclidean rectangle, is properly seen as a projective quadrilateral without metric properties.

One might argue that *how* this quadrilateral is *sampled*, *is* metric dependent. That is, the solid angle subtended by a pixel might be different according to the metric. But in fact this is not so. The solid angles can be thought of as determined by tangent vectors belonging to the point $(0,0,0,1)$ (the canonical position of the camera), and the tangent space of vectors at this point is metrically identical in all three metrics! Hence we don't need to do sample any differently when rendering in a noneuclidean scene.

A confirmation of the projective nature of the camera is provided by practical experiences with clipping planes. Normally, one considers the near and far clipping planes as defined by two positive z-values $0 < z_n < z_f$. But to clip effectively in elliptic space one must use affine coordinates on the z-axis, which includes the value $\infty$, where the z-coordinate shifts to be large and

negative. Typical values for elliptic clipping planes are then $z_f = -z_n$. The resulting viewing frustum includes the plane $w = 0$, the *equator* of elliptic space, and continues almost to the antipodal point of the camera position. [4] See [Wee02] for details.

## 3.3 Visualization issues with elliptic space

As described above, the hardware layer of today's GPU is designed to handle the *standard* homogeneous coordinates needed for euclidean rendering. For this reason, one has slight reason to expect then that $\mathbf{S}^n$ would be faithfully implemented in hardware. However, due to a technical detail in how clipping to normalized device coordinates is implemented, it is indeed possible with a little extra work to represent and render $\mathbf{S}^n$ correctly also. To be precise, at this point in the rendering pipeline, half of $\mathbf{S}^3$ (where $w < 0$) is clipped away. By rendering the scene twice, once after transforming by the negative identity matrix $-\mathbf{1}$, one gets a correct, complete rendering of $\mathbf{S}^3$. For more details see [Wee02].

Being able to render spherical space is a mixed blessing, since it means one also has to expend a little extra work to render correctly in elliptic space. Elliptic points for which $w < 0$ as described above, will be clipped away. Even if your coordinates are good, if you've written an elliptic fly tool with a natural parametrization of an elliptic line using circular functions[5] , then half the time you'll probably be flying in the $w < 0$ hemisphere and won't see anything either. To avoid these clipping problems, one solution is to adjust the scene graph to include two copies of the scene, one transformed by the identity matrix $\mathbf{1}$, the other transformed by $-\mathbf{1}$. This guarantees that one complete copy of the world will be in the $w > 0$ hemisphere and will be rendered. Elliptically the two copies are identical so correct images will be rendered.

Related problems of this nature arise when drawing line segments. One of the oddities of projective space is that two points determine not one but two line segments along the line. Viewed with euclidean lenses, one of these is finite and the other contains the ideal point of the line, so its easy to tell the difference and avoid the problem in euclidean space. The problem doesn't appear in hyperbolic space either, for the same reason. But in $\mathbf{Ell}^3$, there are always two possible line segments between two points. The situation is complicated by the w-clipping issue described above. The proper solution to this dilemma is a topic of current research.

[4] OpenGL correctly implements such clipping planes but other rendering systems display a euclidean bias here.

[5] That is, flying along the z-axis using the parametrization $(0,0,\sin(t),\cos(t))$ for this line).

# 4 METRIC-NEUTRAL INFRASTRUCTURE

This and the following section are intended to serve as a practical guide for progammers interested in extending conventional visualization software to be metric-neutral. In this section we describe changes which have to be made without taking user interaction into account. We term this the *infrastructure* layer. The next section focuses on ensuring metric-neutral user interactions, including picking. We term this the *interaction* layer. Finally, we devote a third section to the challenges of metric-neutral visualization in immersive environments: the *immersive* layer.

## 4.1 Infrastructure challenges

There are a number of areas where the implicit euclidean bias of visualization software makes itself felt. Typically the problems are of two sorts: either one can directly generalize a given euclidean feature (for example, distance between points); or one cannot (for example, free vectors in euclidean space; or, similarity transformations in euclidean space). We term the former a metric-neutral feature, and the latter, a *metric-specific* feature. There are also metric-specific features of elliptic and hyperbolic space but they lie outside the scope of this introductory treatment. For a metric-specific feature, one must then design a solution where the feature is maintained as a special case.

Accompanying this discussion, we present a reference implementation which presents a metric-neutral solution for each of the identified problem areas. This software framework is jReality, an open-source, 3D Java scene graph [WGH+09]. We restrict our discussion here to the jReality features relevant to metric neutrality; see the jReality web-site and Wiki for further documentation for the software, including a tutorial example illustrating noneuclidean usage.

This discussion can not aim to be exhaustive given the great variety of modern visualization systems. Our aim here is to give an overview and make a convincing case that most – if not all –euclidean infrastructure can be extended to be metric-neutral by following a simple set of patterns.

**Geometric representation** The system needs to support homogeneous coordinates for points while maintaining backward compatibility with non-homogeneous representations. Certain geometric entities, such as free vectors, are euclidean metric-specific.

The core space for jReality is $\mathbf{R}P^3$, not $\mathbf{R}^3$. jReality also supports nonhomogeneous coordinates for traditional applications; users interested in noneuclidean geometry will work with homogeneous coordinates for points, normals, and other geometric entities. Points are promoted to be homogeneous (by appending a 1) when operations on mixed types are requested. A free vector

$(x,y,z)$ is handled by converting it into homogeneous coordinates as the ideal point $(x,y,z,0)$.

**Geometric operations** There are a subset of operations which are purely projective, such as the join and meet operators. Implementations of these operations however often do not handle the case of parallel elements in a projective way, hence often lead to incorrect results when used with other metrics. Many other operations based on geometric primitives depend on the ambient metric. For example: distances between points, angles between planes, normal vector to a plane, inner product of two vectors, and orthogonal complement and projection.

Modeling operations based on such primitive operations must also be considered. For example, consider a tube around a line segment. A tube is an equidistant surface – the set of points a given distance from a line segment. In euclidean space, such an equidistant surface is a cylinder, but in the other metric spaces, tubes take analogous but different forms[6].

In jReality, purely projective operations are implemented within $\mathbf{R}P^3$. The intersection point of three planes, for example, is calculated correctly even if two of the planes happen to be euclidean parallel. Subsequent metric operations can signal errors if these projective values are not metrically valid. Additionally, all the metric operations mentioned above plus many others are available in metric-neutral form. Implementation details can be found below (Section 4.2). The standard tubing option for the default jReality line shader uses such built-in features to create (for the first time) accurate noneuclidean tubes around polylines in noneuclidean space. See Figure 2.

**Isometries** Scene graphs are typically built up by applying a transformation at each node in the graph. Except in the case of the camera node, where a perspective transformation is applied, these transformations are typically either euclidean isometries or isotropic scaling operations. A metric-neutral system must provide support for generating noneuclidean isometries in place of the euclidean ones. The user himself must be careful when applying scaling transformations. In general, scaling can only be applied in a metric-neutral way to the leaves of the scene graph.

jReality includes support for calculating projectivities in $PGL(n+1,\mathbf{R})$ including: central projections, harmonic homologies, affine transformations (including scales); and metric isometries including translations, rotations, reflections and glide-reflections, and screw motions (in all three metrics). Factorization of isometries is also supported.

[6] Note that spheres (equidistant surfaces to a point) do not provide the same problem, since a noneuclidean sphere centered at the origin $(0,0,0,1)$ is also a euclidean sphere, and this sphere can be translated to an arbitrary center using a noneuclidean isometry.

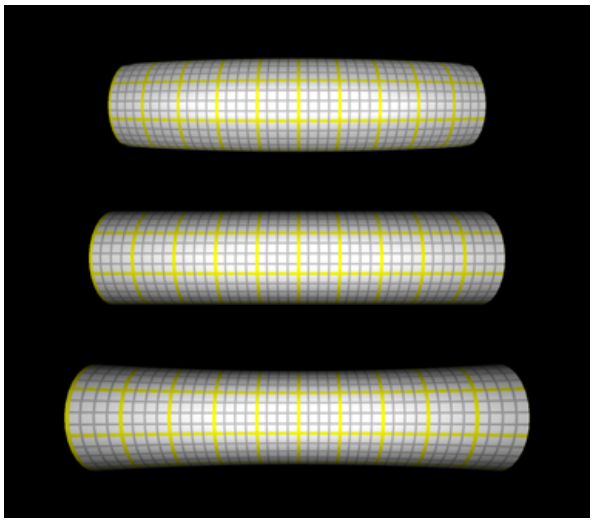


Figure 2: Hyperbolic, euclidean, and elliptic tubes around a horizontal line segment

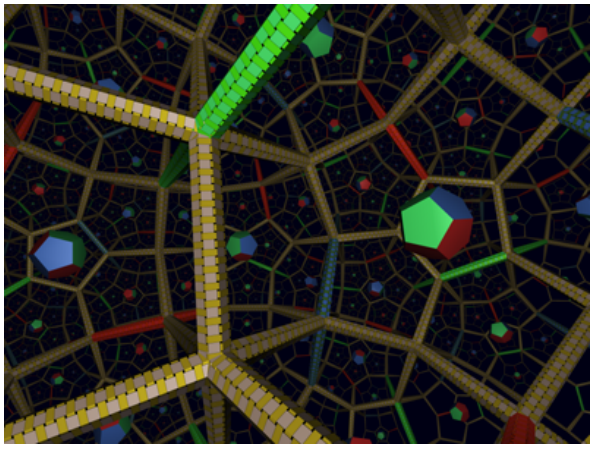


Figure 3: Real-time hyperbolic shading implemented with an OpenGL Shading Language vertex shader

**Shading** Standard real-time shading algorithms are local calculations based on the geometric and material properties of the object and the position and attributes of the light sources. All these properties are well-defined in the noneuclidean setting also.

jReality includes a GPU vertex shader (written in the OpenGL Shading Language) which extends a standard euclidean polygon shader to handle noneuclidean metrics. It operates with homogeneous coordinates for points and normals as provided by jReality, and calculates distances and angles using the appropriate inner product. It also implements light attenuation and fog in a noneuclidean fashion. This shader is similar to the Renderman shader described in [Gun93]. The result is the first real-time realistic rendering integrated into a metric-neutral visualization system. See Figure 3.

**3D Audio** Although technically not part of *visualization* systems, spatial audio can also be implemented in a metric-neutral way. Euclidean biases in spatial audio express themselves in amplitudes, delays, echoes,

and other effects involving distance and angle measurement.

jReality supports noneuclidean spatial audio. All distances required for audio effects, such as the Doppler effect, are calculated in a metric-neutral fashion.

## 4.2 Implementation details

jReality uses a flexible attribute inheritance mechanism within the scene graph to define an attribute which takes one of three values corresponding to hyperbolic, euclidean, or elliptic. Any operation defined within the scene graph is carried out using the current value of this attribute. Through this mechanism it is possible to mix metrics in the scene graph. For example, one could model a mathematical museum in our ordinary euclidean space that includes a non-euclidean exhibit (as a subgraph). See also the discussion of 3D GUI below (Section 6.6). Note that the metric is attached to the parent scene graph component rather than to the geometry node itself, which can be considered as a projective object modified by the enclosing metric attribute.

The mathematical infrastructure in jReality is organized via a set of Java classes which offer functionality via static methods. The principle functional classes are `Rn` and `Pn`, corresponding to the euclidean vector space $\mathbf{R}^n$ and real projective space $\mathbf{R}P^n$, *resp*. Roughly speaking, `Rn` expects nonhomogeneous coordinates for geometric entities; `Pn` on the other hand is based upon homogeneous coordinates. Within `Pn` there are two types of methods: purely projective ones, and metric-neutral ones, parametrized by the metric. The class `P5` (representing 5-dimensional real projective space) provides metric-neutral methods for calculating with lines using Plücker coordinates, see [Kle27].

## 4.3 Geometric algebra

We have embarked upon a project to upgrade the metric-neutral infrastructure to be based on the 3D homogeneous model of geometric algebra ([DFM07], [Sel05]). This has the advantage that it handles operators and operands in a single unified form with built-in metric neutrality. For example, if $m$ is the element of the geometric algebra representing a line, and $X$ is an element representing a point, line or plane, then the rotation of $X$ around the line $m$ by angle $2\alpha$ can be written as a versor, or *sandwich* operator:

$$X \to e^{\alpha m} X e^{-\alpha m}$$

where juxtaposition of elements represents the geometric product of the geometric algebra, which also expresses the metric relations. Readers familiar with the quaternion calculus for representing rotations around the origin in $\mathbf{R}^3$ should recognize a similarity which is more than coincidental. This approach promises to be the right form for handling kinematics and dynamics, too. For an account of the current state of this work see [Gun10].

# 5 METRIC-NEUTRAL INTERACTION

In this section we turn our attention to how human movements originating in a euclidean world can be used to control interaction with a virtual noneuclidean world. We first describe how picking is handled, before turning to tool construction.

## 5.1 Metric-neutral Picking

Picking is generally understood as finding the intersections of a ray with the objects in the scene, sorted in increasing distance from the origin of the ray. To express this operation in a metric-neutral way, one must first replace the ray (a euclidean concept) with a projective line segment $[P_s, P_e]$, typically the intersection of an oriented line with the viewing frustum. Since two points on a projective line determine two segments, one must take care segment is intended. Furthermore, one cannot, in general, use the euclidean distance along this segment as the sorting key. And, since jReality allows different metrics to coexist in the same scene graph, it's also not possible to replace the euclidean distance by some other single metric distance.

Instead, jReality implements picking by calculating, for each intersection, an *affine* coordinate along the pick segment which can be used for sorting purposes. First, it calculates $(u,v)$ barycentric coordinates of the hit point $P$ with respect to the start and end points along the segment: $P = uP_s + vP_e$. The homogeneous representatives for $P_s$ and $P_e$ must be chosen so the signs of $u$ and $v$ are the same for points lying on the segment. Then it uses the ratio $\alpha = \frac{v}{u}$ as the affine coordinate; $\alpha$ takes the value 0 at $P_s$, $+\infty$ at $P_e$, and runs through all positive values in between. Negative values correspond to hits which lie outside the pick segment.

## 5.2 Mouse-based tools

Standard desktop interaction occurs via a 2D motion of a mouse or stylus. A series of 2D points are fed as input to an interactive tool which uses them to generate a 3D motion: rotation or dragging of selected parts of the scene or flying through the scene, for example. For example, when dragging an object, the point of the object under the cursor when the mouse is depressed, remains under the cursor as the cursor is moved, and remains in the same plane parallel to the viewport plane of the camera. Similar but less direct correspondences apply to rotation and scaling tools.

Most, if not all, such interactive tools can be easily converted to be metric-neutral. In the dragging example above, the tools behaves identically except that it uses a noneuclidean translation in place of the euclidean one (see Section 2.1). The case of a rotation tool is even simpler. A rotation tool acting on a given object in the scene is typically implemented by conjugating a rotation around the origin of the object coordinate system

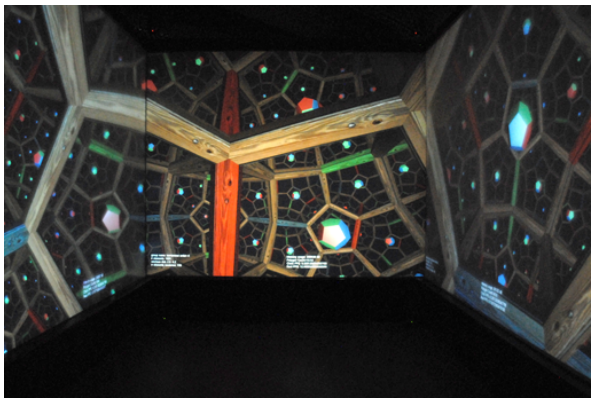


Figure 4: Immersive experience of hyperbolic space in a 3-walled CAVE.

with the *world-to-object* transformation. Since the rotations around the origin are the same in all metrics, such a rotation tool is almost metric-neutral to begin with. Such noneuclidean tools were already included in [MLP+].

# 6 METRIC-NEUTRAL IMMERSIVE ENVIRONMENTS

By an immersive environment we mean an environment featuring 3D glasses, multiple displays, and tracked movement which produce for the user the illusion of being immersed in a virtual world. jReality provides support for such environments via its flexible *backend* concept. Figures 1, 4 show jReality applications running in a CAVE-like theatre. Such environments present special challenges for metric-neutral software. We consider first the challenge presented by generalizing interaction based on a 2D mouse to interaction based on a 3D wand.

## 6.1 Wand-based tools

Instead of using the position of a 2D cursor to determine the picking ray into the scene, the 3D line determined by the wand is used. But since there may be multiple screens, interactive tools cannot depend on a distinguished direction to constrain motion (like the drag tool above which moves the object parallel to the plane of *the* viewport). Since this problem is orthogonal to metric neutrality, we do not handle it further.

Metric-neutral picking, however, does present a problem in immersive environments, since the pick segment cannot simply be chosen as the intersection of the pick segment with *the* viewing frustum: in an immersive environment there may be several such frustums, in each of which valid pick hits can occur. (This problem doesn't arise when picking with a ray since then the frustum doesn't play a role.) The correct solution is to pick in each frustum separately and then combine the picks together. Once these problems have been identified, metric-neutral wand-based tools can be (and in jReality have been) reliably written and used.

## 6.2 Metric-neutral tracking

Perhaps the most important ingredient in virtual reality comes from tracking the real motion of the observer. Tracking systems typically provide a euclidean frame (position and orientation) for each tracked object (for example, head and hand). Each frame is equivalent to a euclidean isometry that moves the standard frame at the origin to the current position and orientation of the object. The illusion of motion in a virtual *euclidean* world is achieved as follows: the left (right) eye, positioned slightly to the left (right) of the origin, is transformed by the tracking isometry to yield its moved position $P$. Then a euclidean translation $T_e$ is used to move a virtual (off-axis) camera from the origin to this position and images are rendered for each display wall, creating the illusion of moving around in the virtual world[7].

In this section we will consider the metric-neutral *tracking* problem: how can the above process be adapted so as to produce the illusion of motion in a *noneuclidean* virtual world?

Previous experiments with noneuclidean virtual reality ([FGK+03]) used this euclidean translation to move around within the noneuclidean scene. This is equivalent to the situation mentioned above in Section 4.2, where a noneuclidean exhibit is viewed by an museum visitor in euclidean space. This can be a useful mode of investigation but it does not represent the insider's view discussed above.

We describe a metric-neutral tracking solution below. The explanation consists of two parts. First we state and solve the so-called scaling problem, and then the tracking problem proper. We then discuss this solution in order to clarify some of the perhaps unfamiliar concepts involved.

## 6.3 Unit lengths and the scaling problem

For the purposes of this discussion, we assume we are dealing with a CAVE-like immersive environment (hereafter referred to as the *room*) shaped like a cube, with the origin of the coordinate system in the middle of the cube.

As mentioned above, we cannot use scaling transformations in a metric-neutral scene graph. So, if the physical dimensions of the tracking system are inappropriate, we can't scale the world to correct this. For example, a tracking system that reports lengths in inches will be inappropriate for viewing hyperbolic space, since the standard model of hyperbolc space will occupy a ball

[7] The orientation matrix for the head does not appear in the scene graph directly, since the images projected on the walls only depend on the *location* of the eye. But it does play a role in determining the position of the eye.

with radius 1 inch. We can however change these dimensions themselves in a metric-neutral way.

This is simple within the jReality tool system. We insert a *virtual device* between the raw tracking device (in meters or inches) and the tools themselves. This virtual device scales the entries of the translation vector by a fixed scale factor; the rotational part is left alone. We can choose the scale factor to *shrink* or *expand* the virtual room so that it occupies the desired subset of the space in question. This flexible unit length is metric-neutral since it avoids inserting scaling transformations into the scene graph itself. It can be carried out in real-time.

Any parameters in the system which depend on the unit length must then be updated when the unit length is changed, for example, the eye separation of the tracked observer.

## 6.4 Metric-neutral tracking

With this flexible coordinate system for the immersive environment in place, it is straightforward to adapt the tracking process to be metric-neutral. First, we inspect the metric attribute of the virtual camera node to determine in which metric the tracking should be done. If it's noneuclidean, construct the unique non-euclidean translation $T_n$ that moves the origin $(0,0,0,1)$ to the same homogeneous point $P$ as $T_e$ does (see Section 2.1) and apply this to the virtual camera instead of $T_e$. All cumulative transformations in the scene graph remain valid noneuclidean isometries, so the images on the walls remain valid views for a noneuclidean insider.

Suppose there is a feature of interest located at $P$. $T_n$ will bring the observer to this point, as $T_e$ does. Furthermore, and in contrast to the use of $T_e$, as one moves nearer to or away from $P$, the feature will appear to increase or decrease in size in a correct noneuclidean way. For example, in hyperbolic space, as the observer backs up away from the front wall, the objects seen on the front wall will tend to reduce their apparent size exponentially quickly. Using a euclidean tracking translation misses this effect. The next section explores this in more depth.

## 6.5 Discussion

Suppose we include in our scene a representation of the room. Call this the *virtual* room. Assume that the illusion of immersion is not complete, and that we can see both the physical room and the virtual room as we move around. For a euclidean observer, the virtual room will coincide with the physical room, if the immersive environment is functioning correctly. What will a noneuclidean observer see?

When standing in the middle of the room, $T_n = T_e$, so the virtual and physical coincide. If the observer backs up until his head is the middle of the back wall, the visual angle subtended by the front wall will depend on

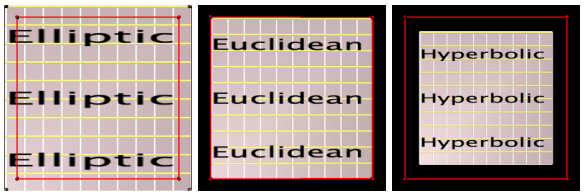



Figure 5: How the front wall appears from the middle of the back wall with a unit length of 1.73m, in the three metrics. Red outline is the physical wall.

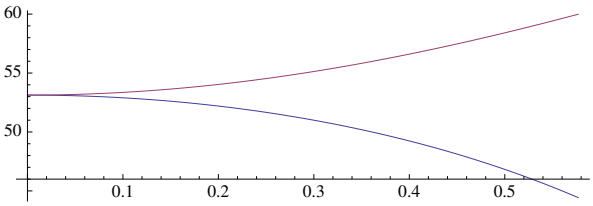



Figure 6: Graph of visual angle in degrees (vertical axis) vs. scaling factor (horizontal). The elliptic case is the red curve; the hyperbolic, blue.

the metric and the unit length chosen. Figure 6 shows the dependence of this visual angle on the scaling factor. The maximum scaling factor on this graph corresponds to a unit length of $\sqrt{3}$, the smallest unit length such that the room contains all of hyperbolic space. Figure 5 compares the view for this maximum value in the three metrics.

A similar effect can be detected in the dihedral angles of the physical walls of the room. An observer carrying a virtual noneuclidean protractor could measure the dihedral angle between the walls of the room. He would discover a deviation from the euclidean right angle. The hyperbolic angle would be less, and the elliptic angle greater.

**The noneuclidean orientation matrix** The foregoing discussion has focused on the translational part of the tracking information, since that is crucial in the head-tracking strategy. What can be said about the orientation part in the noneuclidean setting? This information is important for example in implementing wands and other pointing devices in noneuclidean spaces.

View the orientation part of the euclidean frame as a basis for the tangent space at the origin, and decompose the total frame as $F_e = T_e \circ R_e$ where $R_e$ is the orientation matrix and $T_e$ the euclidean translation (acting as usual on column vectors positioned on the right of the expression). Define a noneuclidean frame $F_n := T_n \circ R_n$, where $T_n$ is as above. Note we can assume $R_n = R_e$ since this is a rotation at the origin, where all three metrics agree.

For a tracked wand, one gets reasonable results by using $F_n$ to transform the scene graph node representing the wand. However, $F_n$ is just one of many possible choices. One might do better by choosing the unique noneuclidean isometry with the same axis and angle of rotation as $F_e$. This will in general be different from

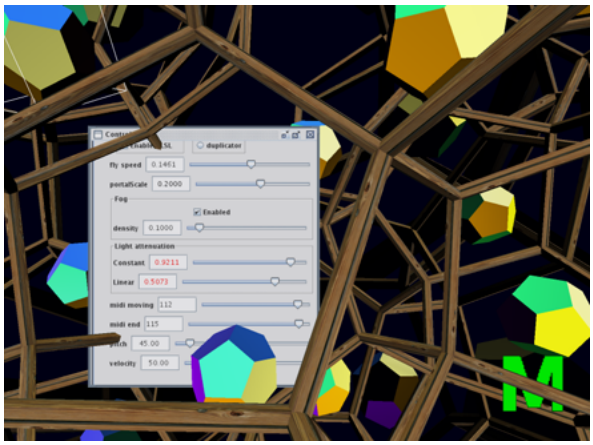


Figure 7: 2D Java GUI embedded as euclidean element in elliptic space.

$F_n$. But in our experience, the difference to $F_n$ is not significant.

### 6.6 Metric-neutral immersive 3D GUI

jReality has the capability to embed standard 2D Java GUI elements (most instances of `java.awt.Component`) as 3D surfaces in the 3D scene graph. See Figure 7. This feature is designed for immersive environments where the standard 2D display surface is absent. The embedding is achieved in a straightforward way by means of texture-mapping and forwarding of input events ([WGH+09]).

We have found that these GUI elements work best in a immersive environment when they are positioned as if they are paintings hung on the walls of the room. The user can then drag and resize these canvases while keeping them on the wall.

## 7 FURTHER WORK

There a no shortage of directions for extending this work. The general program is to identify metric-neutral features and extend the euclidean functionality accordingly, while respecting metric-specific features. The extension to geometric algebra has already been mentioned. Kinematics, rigid body motion, and subdivision surfaces are three further areas of current activity.

## 8 CONCLUSION

We have introduced and characterized a metric-neutral visualization system as one that supports infrastructure, interaction, and immersion for euclidean, hyperbolic and elliptic metrics. We have described a specific implementation fulfilling these requirements, and demonstrated a number of specific innovations, including: metric-neutral tubing, metric-neutral realtime shading, and metric-neutral tracking. The resulting system provides researchers and educators with significant improvement in the quality and ease of visualization of these fundamental mathematical spaces in a metric-neutral context. Furthermore, programming within this system offers an excellent opportunity for students and researchers to deepen their appreciation of the many interesting connections among these three fundamental geometries.

## A CAYLEY-KLEIN METRICS

The following provides the essential knowledge involving construction and calculation of the metric spaces featured in the article. We simplify to dimension $n = 3$. Points in $\mathbf{R}P^3$ are written with homogeneous coordinates as $\mathbf{x} = (x, y, z, w)$.

### A.1 From quadratic form to projective metric

To obtain metric spaces inside $\mathbf{R}P^3$ we begin with a symmetric quadratic form $Q$ on $\mathbf{R}^4$. By standard results of linear algebra we can assume that we have chosen coordinates in which $Q$ takes a diagonal form when expressed as a matrix. The quadric surface associated to $Q$ is then defined to be the points $\{\mathbf{x} \mid \mathbf{x}Q_\varepsilon\mathbf{x}^t = 0\}$. Since $Q$ acts homogeneously on the coordinates, we can consider it also defined on $\mathbf{R}P^3$.

We can use the same matrix $Q$ to define an inner product by interpreting it as a symmetric bilinear form. For our purposes we restrict attention to a special family of $Q$ parametrized by a real parameter $\varepsilon$, and use this to define an inner product between points as follows:

$$\langle \mathbf{x_0}, \mathbf{x_1} \rangle_\varepsilon := \mathbf{x}Q_\varepsilon\mathbf{x}^t = x_0x_1 + y_0y_1 + z_0z_1 + \varepsilon w_0w_1 \quad (1)$$

$\langle , \rangle_1$ gives the inner product for elliptic space, and $\langle , \rangle_{-1}$ for hyperbolic. For brevity we write these two inner products as $\langle , \rangle_+$ and $\langle , \rangle_-$, resp.

The inner products above are defined on the *points* of space; there is an induced inner product on the *planes* of space formally defined as the adjoint of the matrix $Q_\varepsilon$. For $\varepsilon \in \{1, -1\}$ this is identical to the original matrix and we can use the same notation for both points and planes in the formulae below.

The quadric for elliptic space is $\{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{x} \rangle_+ = 0\}$. There are no real solutions; this is called a totally imaginary quadric. The points $\mathbf{x} \mid \langle \mathbf{x}, \mathbf{x} \rangle_+ > 0$ constitute the projective model of elliptic space: all of $\mathbf{R}P^3$. The quadric for hyperbolic space is $\{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{x} \rangle_- = 0\}$, the unit sphere in euclidean space. The points $\mathbf{x} \mid \langle \mathbf{x}, \mathbf{x} \rangle_- < 0$ constitute the projective model of hyperbolic space: the interior of the euclidean unit ball. The sphere itself is sometimes called the *sphere at infinity* for hyperbolic space. The euclidean metric is a limiting case of the above family of metrics as $\varepsilon$ grows larger and larger. In the limit for $\lim_{\varepsilon \to \infty}$, we arrive at the euclidean metric. Here the quadratic form for planes is $diag(1, 1, 1, 0)$, that for points becomes $diag(0, 0, 0, 1)$.The projective model of euclidean space consists of $\mathbf{R}P^3$ with the plane $w = 0$ removed, the so-called *plane at infinity*.

## A.2 Polarity on the metric quadric

A *correlation* of projective space is a projective transformation that maps points to planes; and planes to points. To the absolute quadric $Q_\varepsilon$ is associated a correlation $\Pi$ which maps a point $\mathbf{x}$ to the set $\Pi(\mathbf{x}) := \{\mathbf{y} \mid \mathbf{y}Q_\varepsilon\mathbf{x}^t = 0\}$. When the dimension of $\Pi(\mathbf{x})$ is 2, $\mathbf{x}$ is said to be a regular point. In this case, $\Pi(\mathbf{x})$ is called the *polar plane* of $\mathbf{x}$, and is sometimes written $\mathbf{x}^\perp$, since it is the orthogonal subspace of $\mathbf{x}$ with respect to the metric. The image of a regular plane under $\Pi$ is called its *polar point*. When the quadric is non-degenerate, all points and planes are regular and $\Pi$ is an involution. In the euclidean case, the polar plane of every finite point is the plane at infinity; the polar point of a finite plane is the point at infinity in the normal direction to the plane. Points at infinity and the plane at infinity are not regular and have no polar partner.

The polar plane of a point is important since it can be identified with the tangent space of the point when the metric space is considered as a differential manifold. Many of the peculiarities of euclidean geometry may be elegantly explained due to the degenerate form of the polarity operator.

## A.3 Distance and Angle Formulae

In general a line will have two (possibly imaginary) intersection points $I_1$ and $I_2$ with the absolute quadric. The original definition of distance of two points A and B in these noneuclidean spaces relied on logarithm of the cross ratio of the points $A, B, I_1,$ and, $I_2$ ([Cox65]). By straightforward functional identities these formulae can be brought into alternative form. The distance $d$ between two points $\mathbf{x}$ and $\mathbf{y}$ in the elliptic (resp. hyperbolic) metric is then given by:

$$d = \cos^{-1}\left(\frac{\langle \mathbf{x},\mathbf{y}\rangle_+}{\sqrt{(\langle \mathbf{x},\mathbf{x}\rangle_+\langle \mathbf{y},\mathbf{y}\rangle_+)}}\right)$$

$$d = \cosh^{-1}\left(\frac{-\langle \mathbf{x},\mathbf{y}\rangle_-}{\sqrt{(\langle \mathbf{x},\mathbf{x}\rangle_-\langle \mathbf{y},\mathbf{y}\rangle_-)}}\right)$$

The familiar euclidean distance between two points:

$$d = \sqrt{(x_0-x_1)^2+(y_0-y_1)^2+(z_0-z_1)^2}$$

can be derived by carefully evaluating by parametrizing the above formulas and evaluating the limit as as $\varepsilon \to \infty$. See [Kle27], page 179.

In all three geometries the angle $\alpha$ between two oriented planes $\mathbf{u}$ and $\mathbf{v}$ is given by (where $\langle,\rangle$ represents the appropriate inner product):

$$\alpha = \cos^{-1}\left(\frac{\langle \mathbf{u},\mathbf{v}\rangle}{\sqrt{(\langle \mathbf{u},\mathbf{u}\rangle\langle \mathbf{v},\mathbf{v}\rangle)}}\right)$$

**Guide to literature** To learn more about the mathematics, see [Kle27], [Cox87] and [Cox65], and [Thu97]. For details on how to construct non-euclidean isometries see [PG92], [Gun93], and [Wee02].

# REFERENCES


[Cox65] H.M.S. Coxeter. *Non-Euclidean Geometry*. University of Toronto, Toronto, 1965.

[Cox87] H.M.S. Coxeter. *Projective Geometry*. Springer-Verlag, New York, 1987.

[DFM07] Leo Dorst, Daniel Fontljne, and Stephen Mann. *Geometric Algebra for Computer Science*. Morgan Kaufmann, San Francisco, 2007.

[FGK+03] George Francis, Camille Goudeseune, Henry Kaczmarski, Benjamin Schaeffer, and John M. Sullivan. Alice on the eightfold way: Exploring curved spaces in an enclosed virtual reality theater (cube). In *Visualization and Mathematics III*, pages 304–316. Springer Verlag, 2003.

[GH97] Charles Gunn and Randy Hudson. Mathenautics: Using virtual reality to visit three-dimensional manifolds. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 167–171, Monterey, CA, 1997. ACM.

[Gun92] Charles Gunn. Visualizing hyperbolic geometry. In *Computer Graphics and Mathematics*, pages 299–313. Eurographics, Springer Verlag, 1992.

[Gun93] Charles Gunn. Discrete groups and the visualization of three-dimensional manifolds. In *SIGGRAPH 1993 Proceedings*, pages 255–262. ACM SIGGRAPH, ACM, 1993.

[Gun10] Charles Gunn. Geometric algebra and metric-neutral visualization, kinematics, and dynamics. In *Applications of Geometric Algebra to Computer Science and Engineering*, Amsterdam, 2010. To appear 2011; extended abstract available at `http://www.math.tu-berlin.de/~gunn/Documents/Papers/ga2010-02.pdf`.

[Hee83] Patrick Heelan. *Space-Perception and the Philosophy of Science*. University of California Press, 1983.

[jr06] jreality, 2006. `http://www.jreality.de`.

[Kle27] Felix Klein. *Vorlesungen ueber Nichteuklidische Geometrie*. Chelsea, New York, 1927.

[Mac06] Doug MacKenzie. The poncare conjecture – solved. *Science*, 314:1848–1849, 2006.

[MLP+] Tamara Munzner, Stuart Levy, Mark Phillips, Nathaniel Thurston, and Celeste Fowler. Geomview — an interactive viewing program. For Linux PC's. Available via anonymous ftp on the Internet from `geom.umn.edu`.

[Mun98] Tamara Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Computer Graphics and Applications*, 18:18–23, 1998.

[PG92] Mark Phillips and Charles Gunn. Visualizing hyperbolic space: Unusual uses of 4x4 matrices. In *1992 Symposium on Interactive 3D Graphics*, pages 209–214. ACM SIGGRAPH, ACM, 1992.

[Sel05] Jon Selig. *Geometric Fundamentals of Robotics*. Springer, 2005.

[Thu97] William Thurston. *The Geometry and Topology of 3-Manifolds*. Princeton University Press, 1997.

[Wee90] Jeff Weeks. *The Shape of Space*. Dekker, 1990.

[Wee02] Jeff Weeks. Real-time rendering in curved spaces. *IEEE Comput. Graph. Appl.*, 22(6):90–99, 2002.

[WGH+09] S. Weissmann, C. Gunn, T. Hoffmann, P. Brinkmann, and U. Pinkall. jreality: a java library for real-time interactive 3d graphics and audio. In *Proceedings of 17th International ACM Conference on Multimedia 2009*, pages 927–928, (Oct. 19-24, Beijing, China), 2009.

[WSE04] D. Weiskopf, T. Schafhitzel, and T. Ertl. Gpu-based nonlinear ray tracing. *Computer Graphics Forum*, 23(3):625–633, 2004.

# Fast GPU-based image warping and inpainting for frame interpolation

Jakub Rosner[1]
jakub.rosner@joanneum.at

Hannes Fassold[2]
hannes.fassold@joanneum.at

Peter Schallauer[2]
peter.schallauer@joanneum.at

Werner Bailer[2]
werner.bailer@joanneum.at

## ABSTRACT

Frame interpolation (the insertion of artificially generated images in a film sequence) is often used in post production to change the temporal duration of a sequence, e.g. to achieve a slow-motion effect. Most frame interpolation algorithms first calculate the motion field between two neighboring images and scale it appropriately. Afterwards, the images are warped (mapped) with the scaled motion field, and regions to which no source pixel was mapped are filled up (image inpainting). In this paper, we will focus on the latter two steps, the warping of the images and the image inpainting. We present simple and fast algorithms for image warping and inpainting, and discuss their efficient implementation to GPUs, using the NVIDIA CUDA technology. We compare the CPU and corresponding GPU routines and notice a speedup factor of approximately 6 - 10 for image warping and image inpainting. Significantly higher speedups can be expected for the latest NVIDIA GPU generation codenamed *Fermi* due to several architectural improvements (faster atomic operations, L1/L2 cache). When comparing the result images of the CPU and GPU routine visually, practically no difference can be seen.

## Keywords

Image warping, image inpainting, frame interpolation, GPU, CUDA, GPGPU

## 1. INTRODUCTION

Frame interpolation (the insertion of artificially generated frames in a film sequence) is a commonly used method in video and film post production. It can be used for converting a given film sequence to a slow-motion sequence (also known as retiming). Also when doing film restoration, one can replace missing frames, or frames which have been badly damaged, by artificial frames created by frame interpolation.

Typical frame interpolation algorithms operate in the following way (see Figure 1): As a first step, the pixel-wise motion (optical flow) between the two temporally neighboring images of the interpolated image (which is to be calculated) is estimated. A dense motion field is retrieved, whose motion vectors then are scaled linearly according to the desired temporal position of the interpolated image. After that, one neighbor image is warped with the scaled motion field to get the interpolated image. The term *image warping* means that each pixel of the source image is mapped (translated) with its motion vector and written into a destination image. The interpolated image typically has holes, regions in the image to which no source pixel was mapped to. So the last step is to fill these regions with an image inpainting algorithm. One can apply this procedure to both neighbor images and gets two interpolated images, which can be combined to one image e.g. by some sort of blending . In this work, we will focus on the last two steps, image warping and inpainting, and on their efficient implementation on the GPU using the CUDA technology. We will not describe the calculation of the motion field, as there are efficient GPU-based algorithms available (e.g. [Wer09]) which we will take advantage of.



[1] Silesian University of Technology, PhD faculty of Data Mining, Ulica Academicka 16, 44-100 Gliwice, Poland

[2] JOANNEUM RESEARCH, Institute of Information Systems, Steyrergasse 17, 8010 Graz, Austria

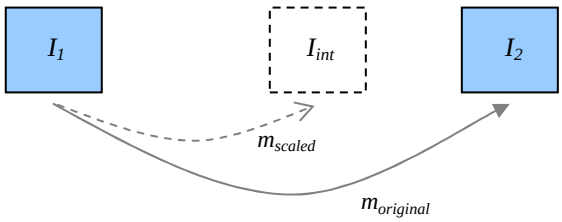



**Figure 1: Illustration of the workflow for frame interpolation. $I_{int}$ is the interpolated image, $I_1$ and $I_2$ its neighbors, $m_{original}$ the calculated motion field between $I_1$ and $I_2$ and $m_{scaled}$ the scaled motion field.**

Device Architecture and is a general-purpose GPU programming environment introduced by NVIDIA, allowing the programmer to utilize the massive processing power of current generation GPUs.

In this document, we first give an introduction into GPU programming and CUDA (see next section). In section 3 we discuss shortly previous work on implementing image warping and image inpainting on the GPU. In section 4 and 5, we give an description of the algorithms we developed in our research group for image warping and image inpainting. After that, in section 6 we describe how we ported our CPU algorithms to CUDA. Finally, in section 7 experiments are done to compare the algorithms and their respective GPU implementations in terms of quality and speed.

## 2. GPU PROGRAMMING & CUDA

In the last years, GPUs have gained significant importance in computer vision and other scientific fields. A number of basic computer vision algorithms has already been implemented efficiently on GPUs, be it optical flow calculation [Wer09], feature point tracking [Fas09] or SIFT features [Sin06]. Typically they provide a speedup of an order of magnitude with respect to a reference CPU implementation, depending on the algorithm's ability to be executed in a massively parallel way. Most GPU implementations use CUDA as it is currently the best supported programming environment.

A CUDA program is typically composed of a control routine, which calls a couple of CUDA *kernels*. A *kernel* is similar to a function, but is executed on the GPU in parallel by a larger number of threads (typically thousands). Groups of 32 consecutive threads are organized into *warps*. Furthermore, sets of up to 512 threads are grouped into *thread blocks*, which then form a *grid*.

An important property of NVIDIA GPUs is *shared memory*, which is a small, but very fast cache which has to be managed by the user. There are also other important memory types with different properties, e.g. texture memory (read-only, cached), constant memory and global memory (read-write, high latency). Also atomic functions are very helpful when different threads try to access the same memory location.

For a more detailed description we refer to the publications [Fas09] and [Ryo08] where GPU/CUDA programming is explained more in depth and guidelines are given for porting algorithms efficiently to CUDA.

## 3. RELATED WORK

Although the literature for image inpainting algorithms is huge (e.g. see [Ber00][Bor07][Che10] [Cri03][Fid08]), there are not many algorithms which have been reported to run on the GPU. This might be because a significant amount of them have a rather complicated workflow or an implicit serial nature which can not be easily mapped to a GPU. In fact, to our knowledge only for one algorithm [Har01] a corresponding GPU implementation has been described[5]. It is implemented in shading language[6], having the disadvantage that the algorithm has to be adapted to fit to the computer-graphics oriented render pipeline. This adaption typically leads to a more complicated implementation and performance degradation. Regarding image warping, a survey of various warping methods can be found in [Wol90].

## 4. IMAGE WARPING

### Algorithm

Image warping is a fundamental task in image processing. Given an source image $I$ and a dense motion field, one wishes to generate a warped image $I_{warped}$ where all the pixels in $I$ have been translated by their corresponding motion vector.

Note that, depending on the motion field, multiple pixels of the source image may map to the same place in the warped image. On the other hand, there may be areas in the warped image to which no source pixel was mapped to, leading to holes in the warped image. Filling up those areas will be described later in this document in section 5.

The algorithm we propose for image warping needs an additional accumulator image and a weight image. Both are floating point (fixed-point is also possible) and are initially set to zero. Now for each source pixel its destination position is calculated, using the mapping defined by the dense motion field. As we

[3] http://www.nvidia.com/object/cuda_home_new.html

[5] http://www.eecs.harvard.edu/~hchong/goodies/inpaint.pdf

[6] http://www.opengl.org/documentation/glsl/

can not write directly to the destination position (it typically has non-integer coordinates), we instead 'write' into the four surrounding pixels of the destination position (one can imagine this as sort of 'bilinear writing'). For that, we *increment* the four surrounding pixels in the accumulator image and also in the weight image. The amount of increment depends on the distance of the destination position to the specific pixel neighbor.

The usage of an accumulator image solves the problem that multiple source pixels possibly map to the same destination pixel. The resultant intensity in the warped image will be a weighted combination of the source pixels intensities.

Finally, the intensity values of the warped image $I_{warped}$ is calculated by dividing the accumulator image pixel-wise by the weight image. Areas to which no source pixel was mapped (holes) are identified by having a zero value in the weight image. A hole mask is generated which is needed for the inpainting process, which is described in the next section. Note that the proposed image warping algorithm is quite fast as it has has linear complexity with respect to the number of image pixels.

## 5. IMAGE INPAINTING

### Algorithm

The input for the image inpainting algorithm is an intensity image *I* and a hole mask *H* which defines the areas of then intensity image, which should be inpainted. In the following, we give an outline of our proposed inpainting method. It needs an additional floating-point accumulator image *A* and weight image *W*. Both are initially set to zero. For multi-channel images, each channel is calculated separately.

First, the set of border pixels of all holes are determined. Now for *each* border pixel, its intensity is *propagated* into the hole in a fixed set of directions (typically 16, equally distributed over the 360 degree range). See Figure 3 for an illustration of the process. The propagation is done in the following way: For a specific border pixel and a specific direction, a line-tracing using the Bresenham algorithm [Bre65] is performed, starting at the border pixel and ending when the line hits the opposite side of the hole. The Bresenham algorithm is slightly modified so that during line-tracing it updates also the approximate distance $d_{curr}$ from the current pixel to the start border pixel. Now, for each visited pixel *p* during line-tracing, its corresponding accumulator image value *A(p)* and weight image value *W(p)* are incremented according to $A(p) = A(p) + \frac{1}{d_{curr}} g_b$ and

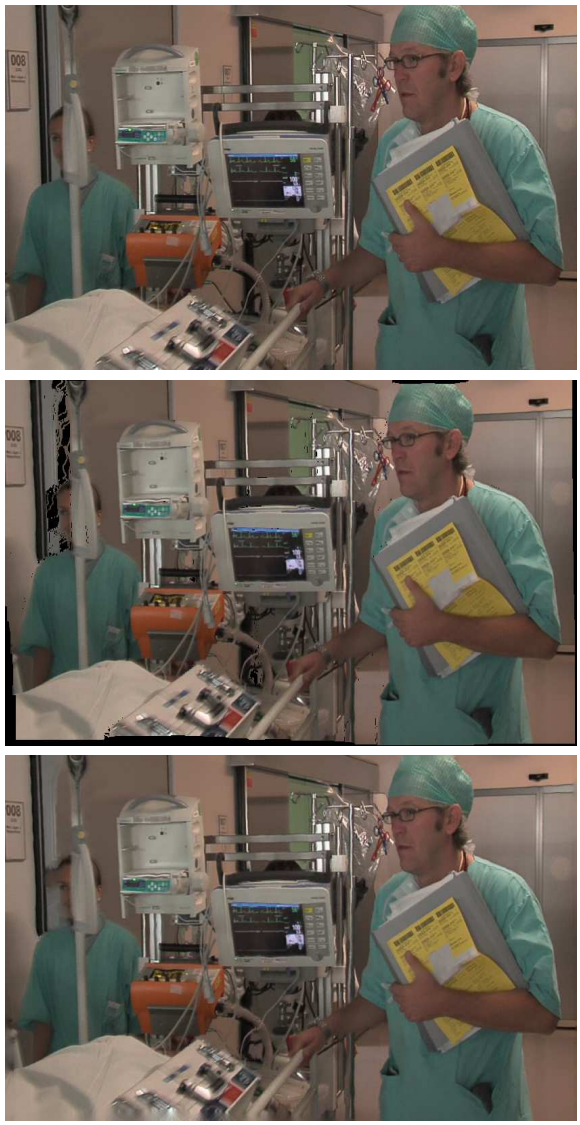


**Figure 2: From top to bottom: input image, warped image, final interpolated image where holes have been inpainted.**

value of the start border pixel. One can see from this that border pixels which are nearer to a given hole pixel have a higher contribution to its intensity value, as the increment in the accumulator image will be higher for them.

After having done the propagation for all hole border pixels and all directions, the intensities values for the regions to be inpainted can be calculated simply by dividing the accumulator image pixel-wise by the weight image.

A problem of the proposed method is that due to using a fixed set of directions, it can introduce star-shaped artefacts into the inpainted regions. In order to reduce these artefacts, we enforce an additional post-processing step.

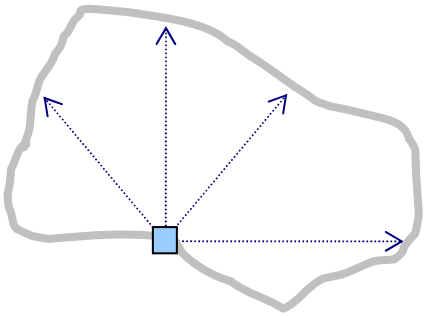


**Figure 3: A specific hole border pixel is propagated into the hole in a fixed set of directions.**

For this purpose, we first calculate a distance map for the hole regions, which gives for each hole pixel approximately its distance to the *nearest* hole border pixel. Note that fast algorithms for calculating the distance map are available [Bor86]. Now each hole pixel is blurred with a distance-adaptive box kernel, with kernel sizes ranging from 3 (for hole pixels near the border) to 9 (inner hole pixels).

One can increase the quality of the inpainted regions by using more directions in the propagation step. On the other hand, also the runtime increases linearly with the number of directions. According to experiments, a value of 16 seems to be a good compromise between quality and runtime.

In Figure 4 the results of proposed image inpainting algorithm (using 16 directions) for some commonly used test images[8] can be seen. As the algorithm is solely diffusion-based, blurring can be observed in the inpainted regions. Note that the proposed algorithm shares some loose similarities with inpainting methods using radial basis functions (RBF) [Uhl06].

## 6. CUDA IMPLEMENTATION

The following section describes some CUDA - specific issues resolved while implementing the image warping and inpainting algorithms for GPU.

The first step is to transfer the source data from CPU memory to GPU memory, unless it already resides in GPU memory. E.g. when the optical flow is calculated with a GPU-based method, then the motion field is already on the GPU and doesn't have to be transferred.

In order to minimize allocation and deallocations of GPU memory, we use a context object which holds the necessary temporary data buffers throughout the whole sequence. The context object can be used for both algorithms (warping and inpainting).

### Image warping

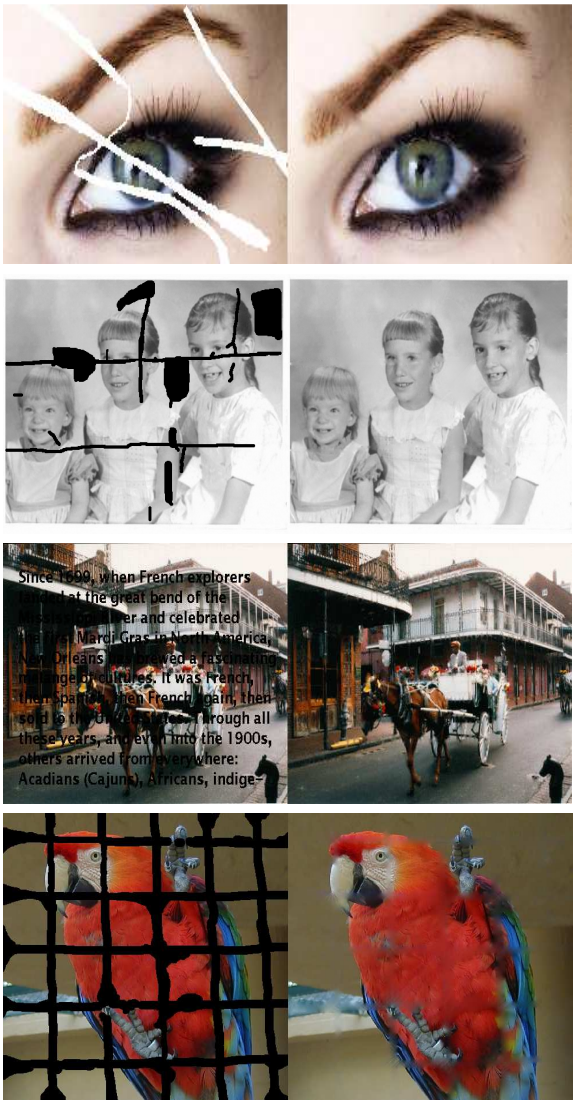


**Figure 4: Image inpainting results for the images Eye, Girls, New Orleans and Parrot.**

The image warping algorithm has been implemented in CUDA in two steps. The first one calculates the accumulator and weight images and the second one then calculates the warped image.

The first step of this algorithm, while being quite straightforward to implement efficiently on the CPU, turns out to be problematic to optimize on graphics processor. The reason is that, multiple GPU threads possibly try to increment the same value in the weight or accumulator images simultanously, leading to read-write hazards. To handle this we have to use *atomic* increment operations which serialize the workflow, but at a large cost in performance.

To reduce this penalty, each thread block (usually 16x16 threads) determines an approximate region in the destination image where its threads will likely be mapped to. As the runtime for executing operations using shared memory is much lower than executing

[8] www-m3.ma.tum.de/bornemann/InpaintingCodeAndData.zip

them using global memory, each thread atomically increments the four pixels around the destination position in global memory only if this position falls outside the approximated region. Otherwise it increments the appropriate values in shared memory, and after all the threads are completed, the whole region is copied into the destination image. Note that the more 'regular' the motion field is, the the more atomic operations in fast shared memory are done.

The final step of the warping algorithm, unlike the first one, is pretty straightforward. It should be noted that for performance reasons, computing the destination value from accumulator and weight image is performed in shared memory.

## Image inpainting

As first step of the image inpainting process, we have to determine the position of all hole border pixels. For this, we first do a 3x3 dilation operation followed by subtraction of the original mask. In the resultant image only hole border pixels have non-zero value. To get a list of their positions, we apply a *compaction* operation which filters out zero-valued pixels. The first two operations are relatively easy to implement on a GPU and a highly efficient compaction algorithm for GPU which was used by us can be found in the CUDA performance primitives (CUDPP) library[9] .

As the next step, the line tracing, involves propagation in different directions, we would encounter cases where multiple threads try to modify the same value at the same time, which would demand the usage of slow atomic functions.In order to avoid this, we split the algorithm into separate kernels, each tracing lines in exactly one direction from each border pixel and therefore prevent multiple-way access hazards. Note that in the line tracing process, each hole border pixel is assigned to one CUDA thread.

To calculate the intensity value for each hole pixel from the accumulator and weight image we use a similar kernel to the one used in image warping, but modify it slightly to compute only the hole regions.

In the last step, the distance-adaptive blurring of the hole regions, for every hole pixel the neighbor pixels for the maximum possible box kernel window size (9x9) are read in, to avoid divergent threads. After that, only the actual needed neighbor pixels (according to the window size for the hole pixel) are used for calculating the result value of the box filter.

# 7. EXPERIMENTS AND RESULTS

In this section we will describe the results from comparing our CUDA implementation against a optimized CPU implementation. The runtime measurements were done on a 3.0 GHz Intel Xeon Quad-Core machine, equipped with a NVIDIA GeForce GTX 285 GPU. The tests have been performed for two commonly used resolutions: Standard Definition (SD) with 720x576 pixels and High Definition 1080p (HD) with 1920x1080 pixels. Note that all the test images are 3-channel color images with 8 bit per channel.

## Quality test

The quality results show that our CUDA implementation of image warping provides the same results in term of quality as the corresponding CPU routine, yet there are some minor differences in the image inpainting. Those differences however are visually indistinguishable and occur only for a small fraction of pixels (on average 8 pixels for SD and 50 pixels for HD have a difference which is higher than a few gray values).

## Runtime test

For doing the runtime comparison, we simulate the frame interpolation scenario. For that, we calculate the motion field between neighboring frames of a short video sequence (10 frames) with the method described in [Wer09] and then do the image warping and inpainting.

In the warped image, on average 1.4 % of the pixels are to be inpainted. For the image inpainting, 16 directions are used. The allocation and deallocation of the context object altogether takes approximately 0.3 milliseconds for SD and 0.6 milliseconds for HD. These times and the time needed for transferring the input image to GPU memory are not included in the given runtime of the GPU implementations. Note that in our application, the input images are already on the GPU as the first step (the calculation of the motion field) was also done on the GPU.

The average speedup (see Figure 5 and Figure 6) which is achieved by the GPU implementations of the algorithms is up to an order of magnitude, clearly demonstrating how advantageous the usage of GPUs can be for sufficiently parallizable algorithms. All runtime numbers are given in milliseconds.

[9] http://www.gpgpu.org/developer/cudpp

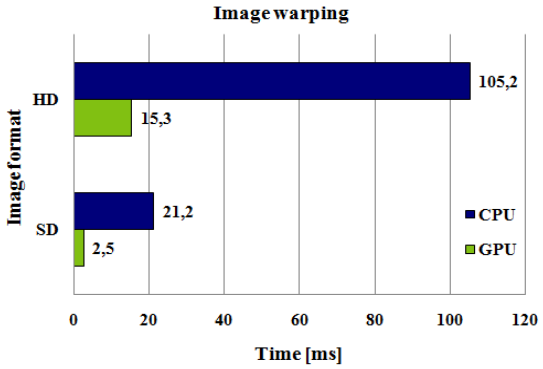



**Figure 5: Runtime of the image warping.**

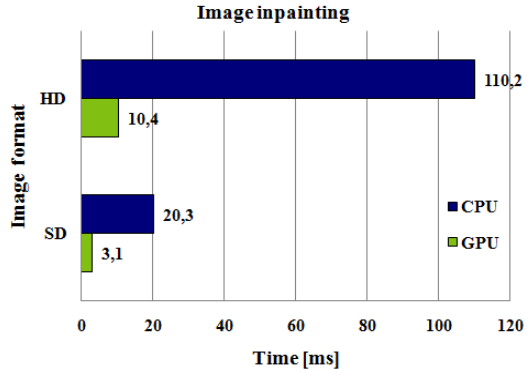



**Figure 6: Runtime of the image inpainting.**

Note that the speedup for image warping is larger for smaller formats, which is counter-intuitive as usually algorithms running on the GPU are more effective for larger sets of data. The reason is that for smaller images the part of image covered by the thread block's shared memory window is relatively larger.

## 8. CONCLUSION

Simple and fast algorithms for image warping and image inpainting for usage in frame interpolation have been presented, and their efficient implementation to the GPU was described. Experiments were done which show an significant speedup factor of 6 – 10 for the GPU implementations of image warping and inpainting. It is expected that in the future this factor keeps the same or even increases as currently GPU generation cycles are shorter than CPU generation cycles.

## 9. ACKNOWLEDGMENTS

This work has been funded partially under the 7th Framework program of the European Union within the project "PrestoPRIME" (FP7-ICT-231161). Furthermore, the work of Jakub Rosner was partially supported by the European Social Fund.

## 10. REFERENCES

[Ber00] M. Bertalmio, G. Sapiro, V. Caselles, C. Ballester, Image inpainting, International Conference on Computer Graphics and Interactive Techniques, 2000

[Bre65] J. E. Bresenham, Algorithm for computer control of a digital plotter, IBM Systems Journal 4, 1965

[Bor86] G. Borgefors, Distance transformations in digital images, Computer Vision, Graphics, and Image Processing, Volume 34, 1986

[Bor07] F. Bornemann, T. März, Fast image inpainting based on coherence transport, Journal of Mathematical Imaging and Vision, Volume 28, 2007

[Che10] X. Chen, F. Xu, Automatic image inpainting by heuristic texture and structure completion, 16th International Multimedia Modeling Conference, 2010

[Cri03] A. Criminisi, P. Perez, K. Toyama, Region filling and object removal by exemplar-based inpainting, IEEE Transactions on Image Processing, Volume 28, No. 9, 2004

[Fas09] H. Fassold, J. Rosner, P. Schallauer, W. Bailer, Realtime KLT Feature Point Tracking for High Definition Video, GravisMa workshop, Plzen, 2009

[Fid08] I. Fidaner, A survey on variational image inpainting, texture synthesis and image completion, Bogazici University, 2008

[Har01] P. Harrison, A non-hierarchical procedure for re-synthesis of complex textures, Proceedings of WSCG, Plzen, 2001

[Ryo08] S. Ryoo, Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, 2008

[Sin06] S. Sinha, J. Frahm, M. Pollefeys, Y. Genc, GPU-Based Video Feature Tracking and Matching, EDGE workshop, 2006

[Uhl06] K. Uhlir, V. Skala, Radial basis function use for the restoration of damaged images, Computational Imaging and Vision, Volume 32, 2006

[Wer09] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, H. Bischof, Anisotropic Huber-L1 Optical Flow, Proceedings of the British Machine Vision Conference, London, UK, 2009

[Wol90] G. Wolberg, Digital Image Warping, IEEE Computer Society Press, 1990

# DirectX 11 Reyes Rendering

Lihan Bin
University of Florida
lbin@cise.ufl.edu

Vineet Goel
AMD/ATI,
Vineet.Goel@amd.com

Jorg Peters
University of Florida
jorg@cise.ufl.edu

**ABSTRACT**

We map reyes-rendering to the GPU by leveraging new features of modern GPUs exposed by the Microsoft DirectX11 API. That is, we show how to accelerate reyes-quality rendering for realistic real-time graphics. In detail, we discuss the implementation of the split-and-dice phase via the Geometry Shader, including bounds on displacement mapping; micropolygon rendering without standard rasterization; and the incorporation of motion blur and camera defocus as *one* pass rather than a multi-pass. Our implementation achieves interactive rendering rates on the Radeon 5000 series.

**Keywords:** reyes, motion blur, camera defocus, DirectX11, real-time , rasterization, displacement

## 1 INTRODUCTION

The reyes (render everything you ever saw) architecture was designed by Pixar [CCC87] in 1987 to provide photo-realistic rendering of complex animated scenes. Pixar's PhotoRealistic RenderMan is an Academy Award-winning offline renderer that implements reyes and is widely used in the film industry. A main feature of the reyes architecture is its adaptive tessellation and micropolygon rendering of higher-order (not linear) surfaces such as Bézier patches, NURBS patches and Catmull-Clark subdivision surfaces. (A micropolygon is a polygon that is expected to project to no more than 1 pixel.)

The contribution of our paper is to judiciously choose features of the Microsoft DirectX11 API to achieve real-time performance. We make heavy, non-standard use of the Geometry Shader and of the tessellation engine, instancing and multiple-render-targets to minimize the number of GPU passes. In particular,

• the Geometry Shader streams out alternatively to a split- or a dice-queue so that all data remains on the GPU,
• we replace triangle rasterization by point rendering,
• we accommodate displacement mapping, and
• we combine blur and de-focus in a single pass followed by a rendering pass to obtain real-time performance. Fig. 1 gives a high-level view of the framework; Fig. 4 provides a more detailed view.

**Overview.** After a brief discussion of prior related work, Section 2 reviews the reyes architecture stages and known efficient algorithmic options. Section 3 shows how our implementation takes specific advantage of the Direct X11 pipeline to obtain real-time performance. Section 4 analyzes the effect of implementation-specific choices and discusses alternatives. We conclude with Section 5 discussing limitations of the present implementation and pointing out how current hardware evolution will impact our framework.



*Brno, Czech Republic.*

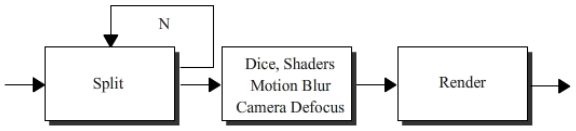




Figure 1: The three types of passes in our framework:the split stage (consisting of *N* adaptive refinement passes), one pre-rendering pass and one final composition pass.

### 1.1 Related Prior Work

The paper assumes basic familiarity with the shader stages of a modern GPU pipeline. In particular, we focus on the MS Direct X11 standard. In *Real-time reyes-style adaptive surface subdivision*, Patney and Owens [PO08] port the split-and-dice stage (see Section 2) on the GPU using CUDA. Micropolygons are rendered via OpenGL so that the approach is too slow for real-time rendering. *RenderAnts* by Kun Zhou et al. [ZHR+09] is an implementation of the complete reyes-pipeline based on a special, newly created, hence non-standard GPGPU programming language, called BSGP. The main contribution of the RenderAnts approach is a software layer which attempts to load-balance within various stages. Our approach uses the hardware to balance the load within different shader stages, avoiding the software overhead. Several recent publications time of this submission) underscore the importance of defocus and motion blur in real-time (non-reyes) settings: *Micropolygon Ray Tracing with Defocus and Motion*

*Blur* [HQL+10] proposes a fast ray traveral data structure for micropolygons, *Real-Time Lens Blur Effects and Focus Control* [LES10] simlutates precise lense effects via (non-micropolygon) ray tracing, accelerated by depth-peeling and [HCOB10] focuses on *Using blur to affect perceived distance and size*. Our approach to motion blur and defocus is not based on ray tracing and different from all three. *Data-Parallel Rasterization of Micropolygons* [FLB+09] proposes an efficient micropolygon rasterization (on the CPU). We adapt this algorithm to the GPU setting and add camera defocus and motion blur as well as displacement mapping in the same pass. We also leverage the simple split-and-dice strategy, *DiagSplit* of Fisher et al. [FFB+09] to reduce the number of micropolygons generated from a diceable patch. However, we base our estimate on a proper bounding box rather than just the corner coefficients.

## 2 THE REYES ARCHITECTURE STAGES AND KNOWN GPU IMPLEMENTATION STRATEGIES.

The reyes architecture stages defined by Cook, Carpenter and Catmull [CCC87] are as follows.

| | |
|---|---|
| Input: | high-order patches |
| Splitting: | Adaptive recursive patch split until the screen space projection is sufficiently small. |
| Dicing: | Uniform split into micropolygons. |
| Shading: | Shaders (Displacement Shader, Surface Shader and Lighting Shader) are applied to each vertex. |
| Sampling: | Multiple samples are generated at different time and different lens positions (using stochastic sampling). |
| Composition: | Samples/Fragments on the same screen location are depth-sorted and blended. |

Below we discuss known good reyes-implementation strategies that influenced our implementation; and we discuss the basics of motion blur and camera defocus.

### 2.1 Splitting

Splitting of the higher-order patch means adaptive subdivision of its domain to partition it into pieces. The goal is to obtain sub-patches that yield micropolygons (that map to one pixel) when uniformly tessellated (diced). The standard split algorithm computes a screen bound of an input patch. If this bound is less than the pixel threshold, the patch is sent to the dice stage where it is tessellated uniformly; otherwise it is split into sub-patches until the condition is satisfied. The algorithm works well when the patch control points are equally spaced. But when the control points are highly non-uniform, then uniform tessellation over-tessellates in some regions or under-tesselates in others.

*DiagSplit* [FFB+09] is a heuristic for avoiding over-tessellation (and taking care of T-joints, where one sub-patch is split but not its neighbor). The idea is to give shorter edges lower tessellation factors and thereby adjust the domain tessellation to the size of the projected image (screen variation). For, if the edge has non-uniform screen variation, uniform domain tessellation in the dicing stage cannot guarantee the micropolygon property: we may get some larger polygons and some extremely small polygons.

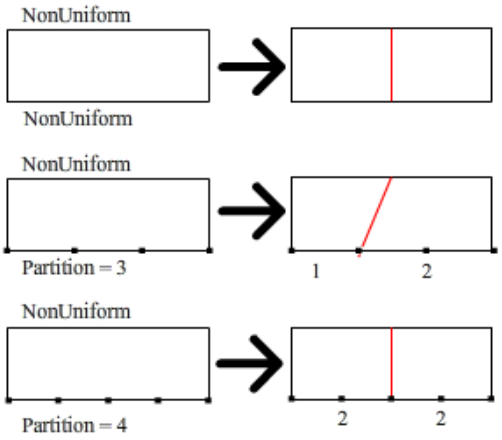



Figure 2: Three DiagSplit scenarios: simple but effective.

In [FFB+09] good correspondence is called *uniform* and poor correspondence *non-uniform*. To check uniformity of domain tessellation with the range tessellation, i.e., uniformity of screen variation under uniform domain tessellation, we step in equal interval through the domain and sample $n+1$ points $p_i$ for each edge. Let $P$ be the projection to screen space, $\ell_i := \|Pp_{i-1} - Pp_i)\|$ and $m := \max_i \ell_i$. If

$$\sigma := |mn - \sum \ell_i| \leq 1 \qquad (1)$$

then $\ell_i \sim m$ for all $i$, i.e. that the range is uniformly partitioned and we have good correspondence. If we then set the edge tessellation factor $\tau$ to $\tau := mn+1$, i.e. partition the domain edge into $mn$ segments, then one domain segment should correspond to one pixel in the range.

Given the edge tessellation factors $\tau_0$ and $\tau_2$ of two opposing edges of a quad, we split the patch according to one of three choices (see Fig. 2).

— If both $\sigma_0 > 1$ and $\sigma_2 > 1$, we split at the mid-points.
— If $\sigma_k > 1$ for only one $k \in \{0,2\}$ then
  — if $\tau_k$ is even, split at the mid-point.
  — if $\tau_k$ is odd, split at $\lceil \tau_k/2 \rceil$.

Watertightness is maintained when shared edges have the same edge tessellation factor.

### 2.2 Motion blur and Camera defocus

Motion blur is an important tool to convey a sense of speed. Long exposure of a moving object creates a

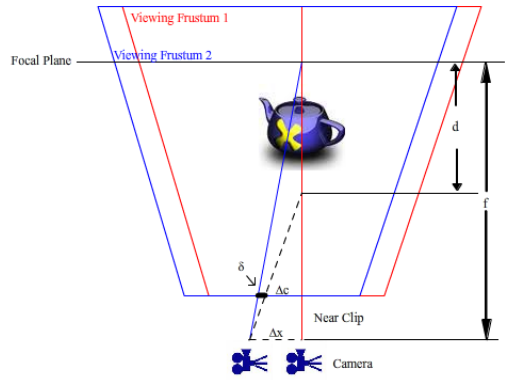



Figure 3: Defocus: simulating defocus by camera shift of less than $\Delta_c$ and projection shift of less than $\Delta_x$ so that $\delta < 1$.

continuous and blurred image on optical film. To approximate the effect of the Model matrix changing from $M_0$ to $M_1$, one averages the images obtained from rendering with $n_{mb}$ intermediate matrices $(1-\frac{i}{n_{mb}+1})M_0 + \frac{i}{n_{mb}+1}M_1$, $i = 1,\ldots,n_{mb}$.

Camera defocus is used to guide viewer attention: as light travels through a lens, only the objects that are within a certain range appear sharp. In order for the light to converge to a single point, its source needs to be in an *in-focus* plane, a certain distance away from the lens. Anything not in or near the plane is mapped to a circle of confusion (CoC) that increases with the distance from the in-focus plane (and the size of the lens). When the CoC is smaller than the film resolution, we call the range in focus and get a sharp image. (In standard rendering, the lens is a 'pinhole', i.e. the lens has zero size yielding always sharpness.)

We can simulate camera defocus, analogous to motion blur, by rendering the scene multiple times. To obtain an un-blurred image within $d$ of the focal point while blurring the remaining scene, we shift the viewing frustum by maximally $\Delta_x$ within a plane (cf. Fig. 3) and adjust the viewing direction towards the focus so that the image of the focal object project to the same 2D screen location. (If the camera shift $\Delta_x$ were too large, only the focal point would be sharp.) That is (c.f. Fig. 3), for a given distance $d_{near}$ to the near clipping plane, focal-region-width $d$ and distance to the in-focus plane $f$, we need to determine upper bounds $\Delta_x$ and $\Delta_c$ by setting $\delta \in \{(\pm 1, \pm 1)\}$, i.e. the allowable perturbations of at most one pixel, We obtain $\Delta_x$ and $\Delta_c$ by solving two linear equations

$$\frac{\Delta_x}{f-d_{near}} = \frac{\Delta_c}{f}, \qquad \frac{\Delta_x - \delta}{f-d_{near}-d} = \frac{\Delta_c}{f-d}$$

arising from similar triangles,

# 3 OUR REYES IMPLEMENTATION.

One update of the image in our implementation uses $N+2$ passes as outlined in Fig. 1 and in more detail in Fig. 4: There are $N$ passes for adaptive refinement (splitting), one heavy pre-rendering pass that combines dicing, shading (computation of normals) and sampling ( displacement, motion blur and defocus ), and a final pass for composition. The implementation uses three buffers: PatchBuffer[0], PatchBuffer[1] and diceablePatchBuffer.

## 3.1 Implementing the reyes-Splitting Stage

In pass 1, in the Geometry Shader, we take PatchBuffer[read] where read = 0 as input and test if a patch is diceable or not. If it is diceable, we stream out to the diceablePatchBuffer, otherwise we stream out to PatchBuffer[write] where write = 1. In the next passes, up to some pass $N$ when the size of PatchBuffer[write] is 0, we satisfy the reyes requirement by repeatedly switching the read and the write PatchBuffer and subdividing the patches. Once the PatchBuffer[write] ends up empty, we switch to the pre-rendering pass.

In more detail, in the split passes, we test each edge of the patch in screen space for uniformity according to (1). Our algorithm takes displacement mapping into account when computing the edge tessellation factor $\tau$. (Displacement mapping moves vertices and thereby changes the size of tessellated polygon violating the micropolygon property. The effect, pixel dropout, is shown in Fig. 9, *top*.) To maintain the micropolygon property, we perform one additional test on each edge: we find the difference between the maximum displacement and minimum displacement, or edge-width, on the edge, as explained in the next paragraph. If the difference is large, we subdivide the edge.

Let the *edge-width* of displacement map be the maximal surface perturbation along a domain edge due to the displacement map. To avoid searching through the displacement map to determine the edge-width, i.e. to avoid a repeated for-loop, we convert the displacement map to a mip-map structure on the GPU. This approach follows the spirit of the CPU algorithm [MM02], in building a mipmap-like structure, but, instead of averaging $2\times 2$ pixels, we compute the *max displacement and the min displacement* values of the $2\times 2$ pixels. This allows us to read off the edge-width for any subdivided patch by looking up the max and min entries at the level of the displacement mip-map corresponding to log(edge-length).

The approach works well, except that it becomes inefficient if the patch has a markedly high-frequency displacement map. Then the upper bound on the edge tessellation factor is high and easily results in overestimation and hence poor performance. We address this by

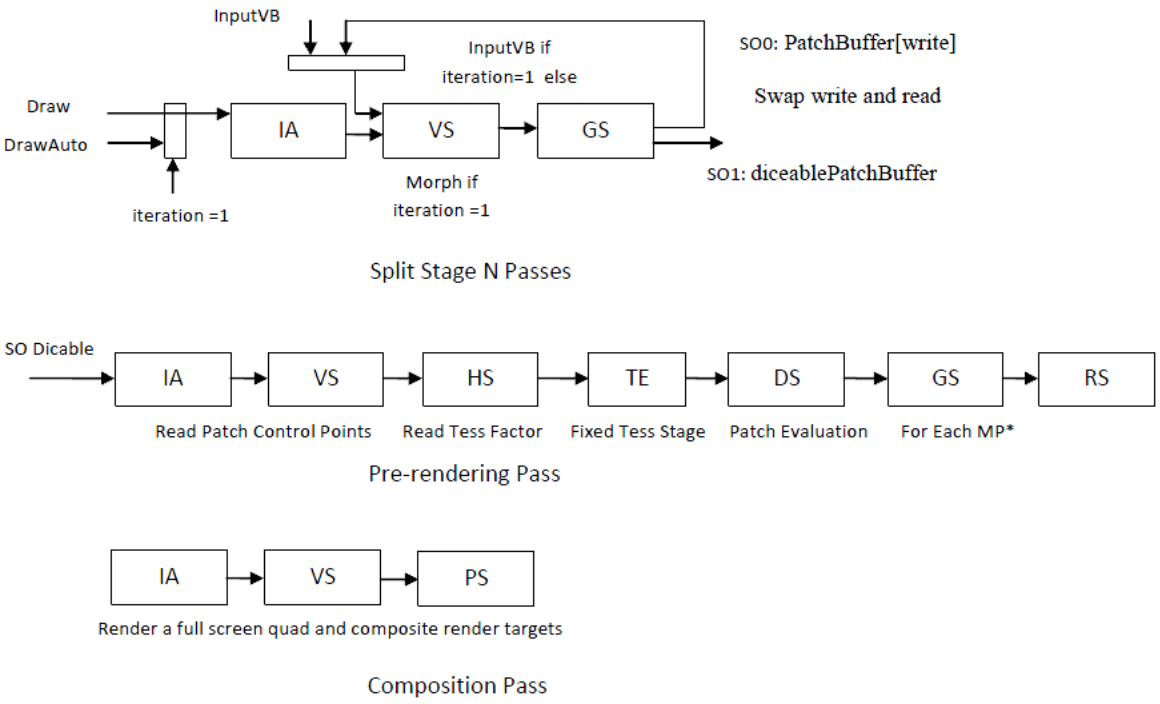



Figure 4: Implementation overview: $N$ initial splitting passes, followed by a heavy pre-rendering pass and a final pass compositing render targets. The pre-rendering pass dices into micropolygons (MP*) using the tessellation engine (Section 3.2); the pass also applies instancing and multiple-render-target in the Geometry Shader to enable defocus and motion blur (Section 3.4); and it computes the micropolygon normal and displacement (with the distortion by the displacement already accounted for into the split stage – see Section 3.1). Abbreviations: IA = Input Assembly, VS = Vertex Shader, HS = Hull Shader, TE = Tessellation Engine, DS = Domain Shader, GS = Geometry Shader, RS = Rasterizer, PS = Pixel Shader, VB= Vertex Buffer, MP=Micropolygon.

clamping to a maximal entry and having the Geometry Shader output sufficiently many points to represent the such unusually big distortion triangles (see Algorithm 1). The result is displayed in Fig. 9, *bottom*.

## 3.2 Implementing the reyes-Dice Stage

We apply the DirectX 11 *tessellation engine* using the edge tessellation factors $\tau$ computed in the Splitting stage. The input to this stage is the diceablePatchBuffer, the output are micro-polygons. Note that this stage, as well as the Shading and the Sampling stages of reyes-rendering are folded into one pre-rendering pass.

## 3.3 Replacing the GPU-Rasterization Stage

With the output of the DirectX 11 Domain Shader stage in the form of micro-triangles of size at most 1 pixel (see Fig. 5), we rasterize in the Geometry Shader. Current hardware rasterization is designed to rasterize *large triangles*. In particular, it can test a 4x4 'stamp' of samples against each polygon to increase throughput and parallelism. However, for each micropolygon this approach streams out at most one pixel wasting all $4 \times 4$ parallelism. Therefore we generate pixel-sized output in the Geometry Shader and send the resulting *point* (not triangle) *per micropolygon* through the hardware rasterizer (which cannot be skipped in current hardware). While the one-primitive-per-clock-cycle rule of the hardware means that the micro-triangle and the point would in principle result in equal speed, using one point increases throughput in the setup, since we send one vertex (point) instead of the three triangle vertices (see Fig. 6).

To determine a correct point sample, we compute the micropolygon's bounding box and then the potentially covered pixels with *findOverlappedPixelGrid*. As explained in Algorithm 1, we then generate a point from any sample that has a successful point-in-triangle test. If all samples fail the point-in-triangle test, there is no need to draw a point since neighboring triangles will provide points for the corresponding pixels. This is akin to Interleaved Sampling [KH01]. We note that without displacement, the projection of any micropolygon is smaller than 1 pixel and then at most four pixels are covered, as illustrated in Fig. 5; but with our correction for high-frequency displacement, more pixels can be generated.

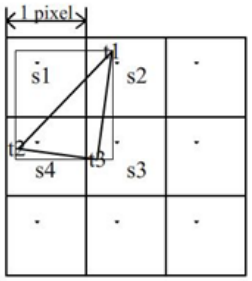



Figure 5: Pixel-footprint of a micropolygon returned by findOverlappedPixelGrid applied to its bounding box. The pointInTriangle test will output $s4$.

**input** : float4[3] pos4D; – triangle vertices
**Uses:** pointInTriangle(q,T,u,v) – true if q is in the triangle T; returns corresponding u,v
**output**: Point $p$

pos2D[0..2] = ProjectToScreen(pos4D[0..2]);
BoundingBox bbox = computeBoundingBox(pos2D);
*PixelGrid* = findOverlappedPixelGrid(bbox);
**for** $q \in$ *PixelGrid* **do**
  **if** *pointInTriangle(q,pos2D,u,v)* **then**
    $p \leftarrow$ pos4D,u,v;
    sendToRasterizer($p$);
  **end**
**end**

**Algorithm 1**: Compute correct point sample as alternative to rasterization

## 3.4 Implementing the reyes-Sampling Stage

The sample stage typically requires one pass per sample for a total of $n$ passes. We leverage the Multiple Render Target call to generate $n$ samples in one pass so that our sampling and composition requires only two passes: a pre-rendering pass and a final composition pass. We use the DirectX 11 OutputMerge stage for the z-test and for alpha-blending. (As usual, for transparency, patches need to be kept ordered in the split stage.)

For camera defocus, we shift the camera by at most $\Delta_c$ and the near-clip-plane by at most $\Delta_x$, projecting with step size $\Delta_c/n_{cd}$, respectively $\Delta_x/n_{cd}$ around the initial position. (This yields $(2n_{cd}+1)(2n_{cd}+1)$ projections). Here $n_{cd}$ is a user-specified number with larger $n_{cd}$ providing better results at higher cost. To combine an $n_{mb}$-transformation motion blur with a $n_{cd}$-sampling camera defocus, we, off-hand, have to render a frame $n_{mb}n_{cd}$ times as shown in Algorithm 2.

However, instancing and the DX11 multiple-render-target call allow us to send objects with different matrices to different render targets. We leverage this to approximate motion blur and defocus *in one pass* in place

**for** $i$ = *0 to* $n_{mb}$ **do**
  **for** $j$ = *0 to* $n_{cd}$ **do**
    Model View Projection Matrix = Model[i] * View[j] * Projection[j]
  **end**
**end**

**Algorithm 2**: Standard algorithm of complexity $n_{mb}n_{cd}$. Our implementation approximates it by a single pass using multiple-render-targets.

of $n_{mb}n_{cd}$ passes (see Fig. 8). This is followed by a final composition pass that blends the entries to generate the image. To meet current limitations of the number of render targets, we set $n_{cd} = 1$ generating 8 different Model matrices (leaving out the unperturbed projection) by calling 8 instances of the Geometry Shader and having the $i$th instance render to the $i$th render target. The composition pass then combines these targets.

Although, for interactive motion blur, we may assume small changes in the Model matrix, there may be triangles whose projection is stretched to violate the micropolygon certification. Just as with extreme displacement (Section 3.1, last paragraph) we then have the Geometry Shader output multiple pointis.

## 3.5 Shadows

To obtain interactive shadows (see e.g. Fig. 8), we interleave passes computing a *shadow map* by reyes rendering from the light position. Here we do not need the all reyes stages but only split, dice and sample; there is no need to shade and composite. By varying the size of the micropolygons, the artist can adjust the shadow quality.

# 4 ANALYSIS OF THE IMPLEMENTATION

Due to hardware limitations, the choice of the edge tessellation factor $\tau$ in Equation (1) (which that determines the number of segments on each edge in the tessellation engine) cannot be arbitrarily large. In fact, in our implementation, we set a *diceable threshold* $\bar{\tau}$ as an upper bound on $\tau$ since we can improve throughput by setting it lower than the current hardware tessellation bound of 64. Table 1 shows how the number of sub-patches varies with $\bar{\tau}$ while the number of evaluation points remains within the same order of magnitude due to adaptive tessellation in the splitting stage. The threshold influences the best load balancing strategy since a lower $\bar{\tau}$ results in a larger number of small patches with low tessellation factor. We tested five different $\bar{\tau}$ values: 8,12,16,20 and 24 (higher values were not competitive). Our implementation achieves the best performance for $\bar{\tau} = 24$.

| $\bar{\tau}$ | tessellation vertices | number of patches |
|---|---|---|
| 8 | 3446260 | 37134 |
| 12 | 2932396 | 15251 |
| 16 | 2763775 | 9378 |
| 20 | 2602425 | 6151 |
| 24 | 2528523 | 3835 |

Table 1: Influence of the diceable threshold $\bar{\tau}$ on the number of patches.

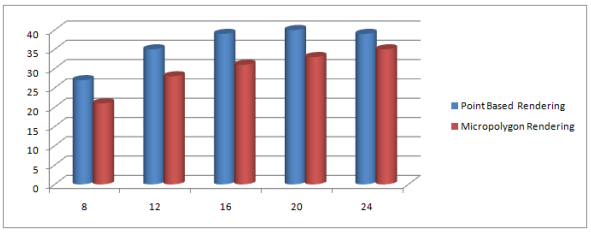



Figure 6: Point-based vs Micropolygon rendering. Performance for a diceable threshold $\bar{\tau} \in \{8, 12, 16, 20, 24\}$ (y-axis:fps, x-axis: $\bar{\tau}$).

As a canonical test scene, we chose the scene in Fig. 9, i.e. a teapot with displacement map and a resolution of $1280 \times 1024$ pixels. Fig. 6 shows that rendering a point is consistently superior to rendering the micropolygon via the rasterizer. Adding motion blur and defocus as another Geometry Shader operation in the pre-rendering pass affects the performance *sub-linearly* in the number of projections: projecting 8 times only reduces the frames per second to one fourth rather than one eighth. We conclude that the pre-rendering pass is not compute-bound. That is, rendering could be faster were it not for the limitation of current GPU hardware to process one primitive per clock cycle: if more than one point could be processed per cycle downstream from the Geometry Shader, the throughput would increase correspondingly. (The simple solution of rendering a polygon the size of several micropolygons would not match the stringent requirements of reyes-rendering.) The main bottleneck, however, is the splitting stage rather than the heavy pre-rendering pass. Fig. 7 shows the extra cost of multiple passes for splitting by juxtaposing the (correct) split+dice performance with a dice-only performance that generates roughly the same number of polgons (but may result in incorrect pixel-dropout).

## 5 DISCUSSION AND FUTURE WORK

Since the pre-rendering pass is not compute-bound, we can accommodate more complex surface and lighting shaders. However, we did not invest into our own complex shaders, since we anticipate that Renderman shaders will in the future be compiled directly on the Geometry Shader obsoleting any custom-made reyes shaders.

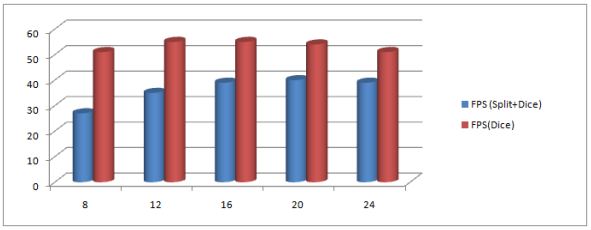



Figure 7: The extra cost of multiple passes for splitting is the difference between correct split+dice and incorrect dice-only (y-axis:fps, x-axis: $\bar{\tau}$).

The proposed framework can accommodate (and is being tested for) order-independent transparency but at the cost of slower performance and additional buffer space, depending on the depth of transparency sorting.

In summary, the contribution of the paper is an efficient use of the DX11 pipeline: split-and-dice via the Geometry Shader, micropolygon rendering without standard rasterization and motion blur and camera defocus as *one* pass rather than a multi-pass via MRT.

## ACKNOWLEDGMENTS

Work supported in part by NSF Grant CCF-0728797 and by ATI/AMD.

## REFERENCES

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 95–102, July 1987.

[FFB+09] Matthew Fisher, Kayvon Fatahalian, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. DiagSplit: parallel, crack-free, adaptive tessellation for micropolygon rendering. *ACM Transactions on Graphics*, 28(5):1–8, December 2009.

[FLB+09] Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. Data-parallel rasterization of micropolygons with defocus and motion blur. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pages 59–68, New York, NY, USA, 2009. ACM.

[HCOB10] Robin Held, Emily Cooper, James O'Brien, and Martin Banks. Using blur to affect perceived distance and size. In *ACM Trans. Graphics*, 2010.

Figure 8: (*left*) Blurring with $n_{mb} = 8$ and (*right*) defocus with $n_{cd} = 1$. See also the accompanying video.

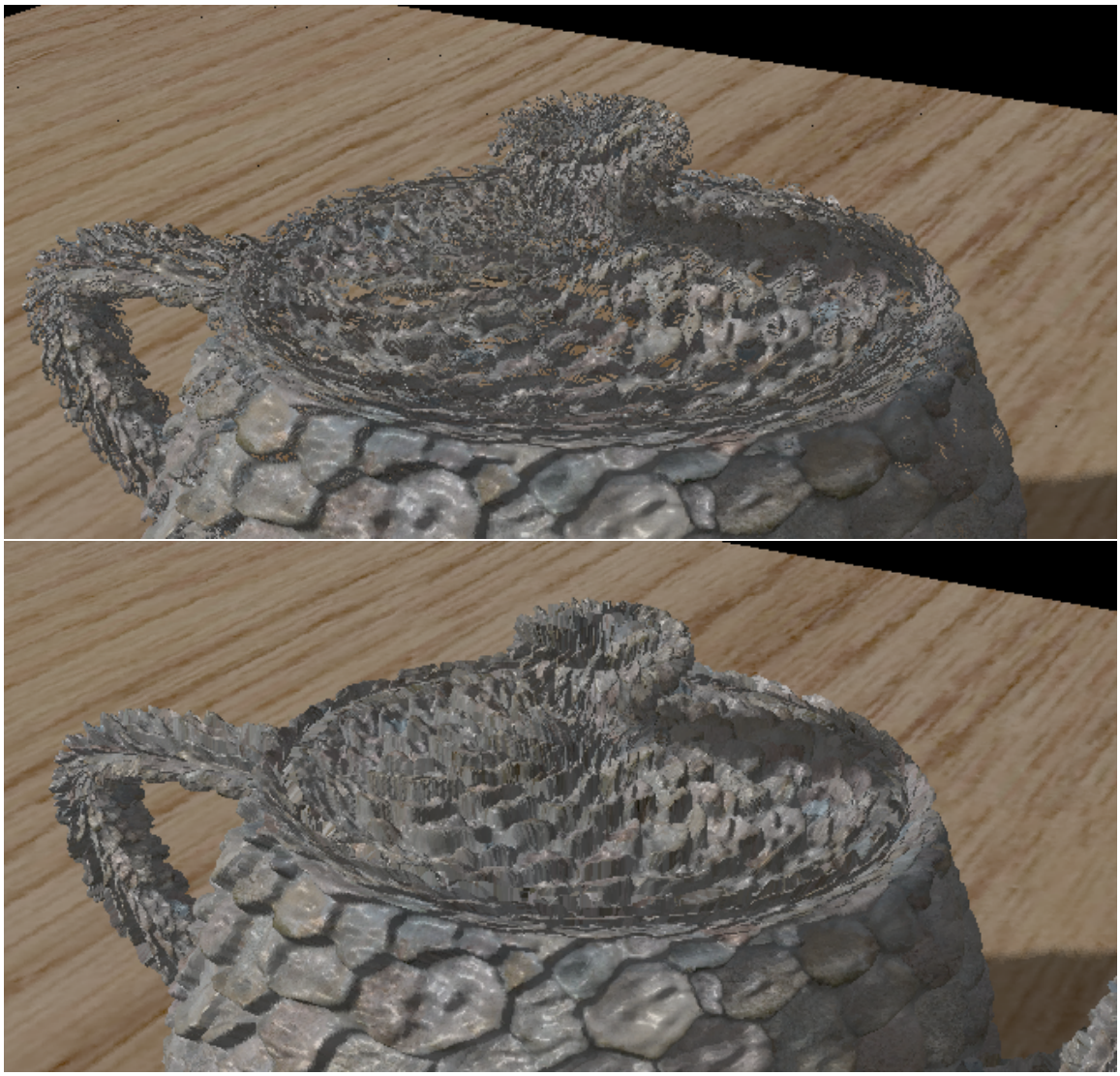


Figure 9: Displacement mapping. (*top*) Pixel dropout due to incorrect treatment with [MM02]. (*bottom*) The problem is fixed by applying the strategy of Section 3.1.

[HQL+10] Qiming Hou, Hao Qin, Wenyao Li, Baining Guo, and Kun Zhou. Micropolygon ray tracing with defocus and motion blur. In *ACM Trans. Graphics, 29(3), 2010 (Proc. ACM SIGGRAPH 2010)*, 2010.

[KH01] Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In Steven J. Gortler and Karol Myszkowski, editors, *Rendering Techniques*, pages 269–276. Springer, 2001.

[LES10] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. In *ACM Trans. Graphics, 29(3), 2010 (Proc. ACM SIGGRAPH 2010)*, 2010.

[MM02] Kevin Moule and Michael D. McCool. Efficient bounded adaptive tessellation of displacement maps. In *Proc. Graphics Interface*, pages 171–180, May 2002.

[PO08] Anjul Patney and John D. Owens. Real-time reyes-style adaptive surface subdivision. *ACM Trans. Graph*, 27(5):143, 2008.

[ZHR+09] Kun Zhou, Qiming Hou, Zhong Ren, Minmin Gong, Xin Sun, and Baining Guo. Renderants: interactive reyes rendering on GPUs. *ACM Trans. Graph*, 28(5), 2009.

# Angles between subspaces

Eckhard Hitzer, Department of Applied Physics, University of Fukui, 910-8507 Japan

August 2, 2010

**Abstract**

We first review the definition of the angle between subspaces and how it is computed using matrix algebra. Then we introduce the Grassmann and Clifford algebra description of subspaces. The geometric product of two subspaces yields the full relative angular information in an explicit manner. We explain and interpret the result of the geometric product of subspaces gaining thus full access to the relative orientation information.

**Keywords:** Clifford geometric algebra, subspaces, relative angle, principal angles, principal vectors.
**AMS Subj. Class.:** 15A66.

## 1 Introduction

I first came across Clifford's geometric algebra in the early 90ies in papers on gauge field theory of gravity by J.S.R. Chisholm, struck by the seamlessly compact, elegant, and geometrically well interpretable expressions for elementary particle fields subject to Einstein's gravity. Later I became familiar with D. Hestenes' excellent modern formulation of geometric algebra, which explicitly shows how the geometric product of two vectors encodes their complete relative orientation in the scalar inner product part (cosine) and in the bivector outer product part (sine).

Geometric algebra can be viewed as an algebra of a vector space and all its subspaces, represented by socalled blades. I therefore often wondered if the geometric product of subspace blades also encodes their complete relative orientation, and how this is done? What is the form of the result, how can it be interpreted and put to further use? I learned more about this problem, when I worked on the conformal representation of points, point pairs, lines, planes, circles ans spheres of three dimensional Euclidean geometry [7] and was able to find one general formula fully expressing the relative orientation of any two of these objects. Yet L. Dorst (Amsterdam) later asked me if this formula could be generalized to any dimension, because by his experience formulas that work dimension independent are *right*. I had no immediate answer, it seemed to complicated to me, having to deal with too many possible cases.

But when I prepared for December 2009 a presentation on neural computation and Clifford algebra, I came across a 1983 paper by Per Ake Wedin on angles between subspaces of finite dimensional inner product spaces [2], which taught me the classical approach. In addition it had a very interesting note on solving the problem, essentially using Grassmann algebra with an additional canonically defined inner product. After that the various bits and pieces came together and began to show the whole picture, the picture which I want to explain in this contribution.



## 2 The angle between two lines

To begin with let us look (see Fig. 1) at two lines $\mathsf{A}, \mathsf{B}$ in a vector space $\mathbb{R}^n$, which are spanned by two (unit) vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$, $\boldsymbol{a} \cdot \boldsymbol{a} = \boldsymbol{b} \cdot \boldsymbol{b} = 1$:

$$\mathsf{A} = \mathrm{span}\{\boldsymbol{a}\}, \quad \mathsf{B} = \mathrm{span}\{\boldsymbol{b}\}. \tag{1}$$

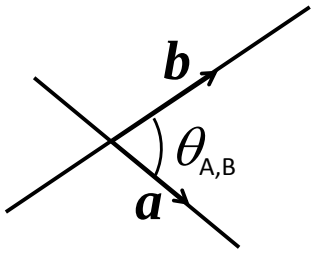



Figure 1: Angle $\theta_{\mathsf{A,B}}$ between two lines A, B, spanned by unit vectors $\boldsymbol{a}$, $\boldsymbol{b}$, respectively.

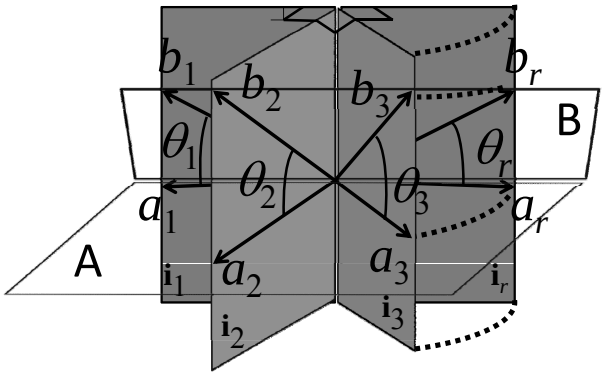



Figure 2: Angular relationship of two subspaces A, B, spanned by two sets of vectors $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r\}$, and $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\}$, respectively.

The angle $0 \leq \theta_{\mathsf{A,B}} \leq \pi/2$ between lines A and B is simply given by

$$\cos\theta_{\mathsf{A,B}} = \boldsymbol{a} \cdot \boldsymbol{b}. \tag{2}$$

# 3 Angles between two subspaces (described by principal vectors)

Next let us examine the case of two $r$-dimensional ($r \leq n$) subspaces $\mathsf{A}, \mathsf{B}$ of an $n$-dimensional Euclidean vector space $\mathbb{R}^n$. The situation is depicted in Fig. 2. Each subspace A, B is spanned by a set of $r$ linearly independent vectors

$$\begin{aligned} \mathsf{A} &= \mathrm{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r\} \subset \mathbb{R}^n, \\ \mathsf{B} &= \mathrm{span}\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\} \subset \mathbb{R}^n. \end{aligned} \tag{3}$$

Using Fig. 2 we introduce the following notation for principal vectors. The angular relationship between the subspaces A, B is characterized by a set of $r$ principal angles $\theta_k, 1 \leq k \leq r$, as indicated in Fig. 2. A principal angle is the angle between two principal vectors $\boldsymbol{a}_k \in \mathsf{A}$ and $\boldsymbol{b}_k \in \mathsf{B}$. The spanning sets of vectors $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r\}$, and $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\}$ can be chosen such that pairs of vectors $\boldsymbol{a}_k, \boldsymbol{b}_k$ either

- agree $\boldsymbol{a}_k = \boldsymbol{b}_k$, $\theta_k = 0$,
- or enclose a finite angle $0 < \theta_k \leq \pi/2$.

In addition the pairs of vectors $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}, 1 \leq k \leq r$ span mutually orthogonal lines (for $\theta_k = 0$) and (principal) planes $\mathbf{i}_k$ (for $0 < \theta_k \leq \pi/2$). These mutually orthogonal planes $\mathbf{i}_k$ are indicated in Fig. 2. Therefore if $\boldsymbol{a}_k \nparallel \boldsymbol{b}_k$ and $\boldsymbol{a}_l \nparallel \boldsymbol{b}_l$ for $1 \leq k \neq l \leq r$, then plane $\mathbf{i}_k$ is orthogonal to plane $\mathbf{i}_l$. The cosines of the socalled principal angles $\theta_k$ may therefore be $\cos\theta_k = 1$ (for $\boldsymbol{a}_k = \boldsymbol{b}_k$), or $\cos\theta_k = 0$ (for $\boldsymbol{a}_k \perp \boldsymbol{b}_k$), or any value $0 < \cos\theta_k < 1$. The total angle between the two subspaces A, B is defined as the product

$$\cos\theta_{\mathsf{A,B}} = \cos\theta_1 \cos\theta_2 \ldots \cos\theta_r. \tag{4}$$

In this definition $\cos\theta_{\mathsf{A,B}}$ will automatically be zero if any pair of principal vectors $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}, 1 \leq k \leq r$ is perpendicular. Then the two subspaces are said to be perpendicular $\mathsf{A} \perp \mathsf{B}$, a familiar notion from three dimensions, where two perpendicular planes A, B share a common line spanned by $\boldsymbol{a}_1 = \boldsymbol{b}_1$, and have two mutually orthogonal principal vectors $\boldsymbol{a}_2 \perp \boldsymbol{b}_2$, which are both in turn orthogonal to the common line vector $\boldsymbol{a}_1$. It is further possible to choose the indexes of the vector pairs $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}, 1 \leq k \leq r$ such that the principle angles $\theta_k$ appear ordered by magnitude

$$\theta_1 \geq \theta_2 \geq \ldots \geq \theta_r. \tag{5}$$

# 4 Matrix algebra computation of angle between subspaces

The conventional method of computing the angle $\theta_{\mathsf{A,B}}$ between two $r$-dimensional subspaces $\mathsf{A}, \mathsf{B} \subset \mathbb{R}^n$ spanned by two sets of vectors $\{\boldsymbol{a}'_1, \boldsymbol{a}'_2, \ldots \boldsymbol{a}'_r\}$ and $\{\boldsymbol{b}'_1, \boldsymbol{b}'_2, \ldots \boldsymbol{b}'_r\}$ is to first arrange these vectors as column vectors into two $n \times r$ matrices

$$M_{\mathsf{A}} = [\boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_r], \quad M_{\mathsf{B}} = [\boldsymbol{b}'_1, \ldots, \boldsymbol{b}'_r]. \tag{6}$$

Then standard matrix algebra methods of QR decomposition and singular value decomposition are applied to obtain

- $r$ pairs of singular unit vectors $\boldsymbol{a}_k, \boldsymbol{b}_k$ and
- $r$ singular values $\sigma_k = \cos\theta_k = \boldsymbol{a}_k \cdot \boldsymbol{b}_k$.

This approach is very computation intensive.

# 5 Even more subtle ways

Per Ake Wedin in his 1983 contribution [2] to a conference on Matrix Pencils entitled *On Angles between Subspaces of a Finite Dimensional Inner Product Space* first carefully treats the above mentioned matrix algebra approach to computing the angle $\theta_{\mathsf{A},\mathsf{B}}$ in great detail and clarity. Towards the end of his paper he dedicates less than one page to mentioning an alternative method starting out with the words: *But there are even more subtle ways to define angle functions.*

There he essentially reviews how $r$-dimensional subspaces $\mathsf{A}, \mathsf{B} \subset \mathbb{R}^n$ can be represented by $r$-vectors (blades) in Grassmann algebra $A, B \in \Lambda(\mathbb{R}^n)$:

$$\begin{aligned} \mathsf{A} &= \{\boldsymbol{x} \in \mathbb{R}^n | x \wedge A = 0\}, \\ \mathsf{B} &= \{\boldsymbol{x} \in \mathbb{R}^n | x \wedge B = 0\}. \end{aligned} \tag{7}$$

The angle $\theta_{\mathsf{A},\mathsf{B}}$ between the two subspaces $\mathsf{A}, \mathsf{B} \in \mathbb{R}^n$ can then be computed in a single step

$$\cos\theta_{\mathsf{A},\mathsf{B}} = \frac{A \cdot \widetilde{B}}{|A||B|} = \cos\theta_1 \cos\theta_2 \ldots \cos\theta_r, \tag{8}$$

where the inner product is canonically defined on the Grassmann algebra $\Lambda(\mathbb{R}^n)$ corresponding to the geometry of $\mathbb{R}^n$. The tilde operation is the reverse operation representing a dimension dependent sign change $\widetilde{B} = (-1)^{\frac{r(r-1)}{2}} B$, and $|A|$ represents the norm of blade $A$, i.e. $|A|^2 = A \cdot \widetilde{A}$, and similarly $|B|^2 = B \cdot \widetilde{B}$. Wedin refers to earlier works of L. Andersson [1] in 1980, and a 1963 paper of Q.K. Lu [5].

Yet equipping a Grassmann algebra $\Lambda(\mathbb{R}^n)$ with a canonical inner product comes close to introducing Clifford's geometric algebra $Cl_n = Cl(\mathbb{R}^n)$. And there is another good reason to do that, as e.g. H. Li explains in his excellent 2008 textbook *Invariant Algebras and Geometric Reasoning* [4]: *... to allow sums of angles to be advanced invariants, the inner-product Grassmann algebra must be extended to the Clifford algebra ...* This is why I have decided to immediately begin in the next section with Clifford's geometric algebra instead of first reviewing inner-product Grassmann algebra.

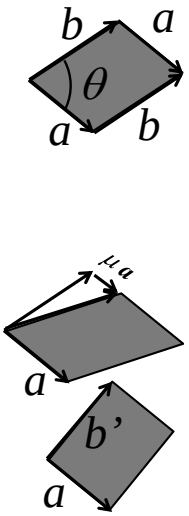



Figure 3: Bivectors $\boldsymbol{a} \wedge \boldsymbol{b}$ as oriented area elements can be reshaped (e.g. by $\boldsymbol{b} \to \boldsymbol{b} + \mu\boldsymbol{a}, \mu \in \mathbb{R}$) without changing their value (area and orientation). The bottom figure shows orthogonal reshaping into the form of an oriented rectangle.

# 6 Clifford (geometric) algebra

Clifford (geometric) algebra is based on the geometric product of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^{p,q}, p+q=n$

$$\boldsymbol{a}\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}, \tag{9}$$

and the associative algebra $Cl_{p,q}$ thus generated with $\mathbb{R}$ and $\mathbb{R}^{p,q}$ as subspaces of $Cl_{p,q}$. $\boldsymbol{a} \cdot \boldsymbol{b}$ is the symmetric inner product of vectors and $\boldsymbol{a} \wedge \boldsymbol{b}$ is Grassmann's outer product of vectors representing the oriented parallelogram area spanned by $\boldsymbol{a}, \boldsymbol{b}$, compare Fig. 3.

As an example we take the Clifford geometric algebra $Cl_3 = Cl_{3,0}$ of three-dimensional (3D) Euclidean space $\mathbb{R}^3 = \mathbb{R}^{3,0}$. $\mathbb{R}^3$ has an orthonormal basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$. $Cl_3$ then has an eight-dimensional basis of

$$\{1, \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{\text{vectors}}, \underbrace{\mathbf{e}_2\mathbf{e}_3, \mathbf{e}_3\mathbf{e}_1, \mathbf{e}_1\mathbf{e}_2}_{\text{area bivectors}}, \underbrace{i = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3}_{\text{volume trivector}} \}. \tag{10}$$

Here $i$ denotes the unit trivector, i.e. the oriented volume of a unit cube, with $i^2 = -1$. The even grade subalgebra $Cl_3^+$ is isomorphic to Hamilton's quaternions $\mathbb{H}$. Therefore elements of $Cl_3^+$ are also called *rotors* (rotation operators), rotating vectors and multivectors of $Cl_3$.

In general $Cl_{p,q}, p+q=n$ is composed of so-called $r$-vector subspaces spanned by the induced bases

$$\{\boldsymbol{e}_{k_1}\boldsymbol{e}_{k_2}\ldots\boldsymbol{e}_{k_r} \mid 1 \le k_1 < k_2 < \ldots < k_r \le n\}, \tag{11}$$

each with dimension $\binom{r}{n}$. The total dimension of the $Cl_{p,q}$ therefore becomes $\sum_{r=0}^{n}\binom{r}{n} = 2^n$.

General elements called *multivectors* $M \in Cl_{p,q}, p+q=n$, have $k$-vector parts $(0 \le k \le n)$: scalar part $Sc(M) = \langle M\rangle = \langle M\rangle_0 = M_0 \in \mathbb{R}$, vector part $\langle M\rangle_1 \in \mathbb{R}^{p,q}$, bi-vector part $\langle M\rangle_2$, ..., and pseudoscalar part $\langle M\rangle_n \in \bigwedge^n \mathbb{R}^{p,q}$

$$M = \sum_{A=1}^{2^n} M_A \boldsymbol{e}_A = \langle M\rangle + \langle M\rangle_1 + \langle M\rangle_2 + \ldots + \langle M\rangle_n . \tag{12}$$

The *reverse* of $M \in Cl_{p,q}$ defined as

$$\widetilde{M} = \sum_{k=0}^{n} (-1)^{\frac{k(k-1)}{2}} \langle M\rangle_k, \tag{13}$$

often replaces complex conjugation and quaternion conjugation. Taking the reverse is equivalent to reversing the order of products ob basis vectors in the basis blades of (11). For example the reverse of the bivector $\boldsymbol{e}_1\boldsymbol{e}_2$ is

$$\widetilde{\boldsymbol{e}_1\boldsymbol{e}_2} = \boldsymbol{e}_2\boldsymbol{e}_1 = -\boldsymbol{e}_1\boldsymbol{e}_2, \tag{14}$$

because only the antisymmetric outer product part $\boldsymbol{e}_2\boldsymbol{e}_1 = \boldsymbol{e}_2 \wedge \boldsymbol{e}_1 = -\boldsymbol{e}_1 \wedge \boldsymbol{e}_2$ is relevant.

The scalar product of two multivectors $M, \widetilde{N} \in Cl_{p,q}$ is defined as

$$M * \widetilde{N} = \langle M\widetilde{N}\rangle = \langle M\widetilde{N}\rangle_0. \tag{15}$$

For $M, \widetilde{N} \in Cl_n = Cl_{n,0}$ we get $M * \widetilde{N} = \sum_A M_A N_A$. The modulus $|M|$ of a multivector $M \in Cl_n$ is defined as

$$|M|^2 = M * \widetilde{M} = \sum_A M_A^2. \tag{16}$$

## 6.1 Subspaces described in geometric algebra

In $Cl_n$ symmetric inner product part of two vectors $\boldsymbol{a} = a_1\boldsymbol{e}_1 + a_2\boldsymbol{e}_2$, $\boldsymbol{b} = b_1\boldsymbol{e}_1 + b_2\boldsymbol{e}_2$ yields the expected result

$$\boldsymbol{a}\cdot\boldsymbol{b} = a_1b_1 + a_2b_2 = |\boldsymbol{a}||\boldsymbol{b}|\cos\theta_{\boldsymbol{a},\boldsymbol{b}}. \tag{17}$$

Whereas the antisymmetric outer product part gives the bivector, which represents the oriented area of the parallelogram spanned by $\boldsymbol{a}$ and $\boldsymbol{b}$

$$\boldsymbol{a}\wedge\boldsymbol{b} = (a_1b_2 - a_2b_1)\boldsymbol{e}_1\boldsymbol{e}_2 = |\boldsymbol{a}||\boldsymbol{b}|\sin\theta_{\boldsymbol{a},\boldsymbol{b}}\boldsymbol{e}_1\boldsymbol{e}_2. \tag{18}$$

The parallelogram has the (signed) scalar area $|\boldsymbol{a}||\boldsymbol{b}|\sin\theta_{\boldsymbol{a},\boldsymbol{b}}$ and its orientation in the space $\mathbb{R}^n$ is given by the oriented unit area bivector $\boldsymbol{e}_1\boldsymbol{e}_2$.

Two non-zero vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are parallel, if and only if $\boldsymbol{a}\wedge\boldsymbol{b} = 0$, i.e. if and only if $\sin\theta_{\boldsymbol{a},\boldsymbol{b}} = 0$

$$\boldsymbol{a}\wedge\boldsymbol{b} = 0 \Leftrightarrow \boldsymbol{a} \parallel \boldsymbol{b} \Leftrightarrow \boldsymbol{b} = \alpha\boldsymbol{a}, \alpha \in \mathbb{R}. \tag{19}$$

We can therefore use the outer product to represent a line $\mathsf{A} = \mathrm{span}\{\boldsymbol{a}\}$ with direction vector $\boldsymbol{a} \in \mathbb{R}^n$ as

$$\mathsf{A} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}\wedge\boldsymbol{a} = 0\}. \tag{20}$$

Moreover, bivectors can be freely reshaped (see Fig. 3), e.g.

$$\boldsymbol{a}\wedge\boldsymbol{b} = \boldsymbol{a}\wedge(\boldsymbol{b} + \mu\boldsymbol{a}), \mu \in \mathbb{R}, \tag{21}$$

because due to the antisymmetry $\boldsymbol{a}\wedge\boldsymbol{a} = 0$. This reshaping allows to (orthogonally) reshape a bivector to rectangular shape

$$\boldsymbol{a}\wedge\boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}', \boldsymbol{a} \perp \boldsymbol{b} (\text{i.e. } \boldsymbol{a}\cdot\boldsymbol{b}' = 0) \tag{22}$$

as indicated in Fig. 3. The shape may even chosen as square or circular, depending on the application in mind.

The total antisymmetry of the trivector $\boldsymbol{x}\wedge\boldsymbol{a}\wedge\boldsymbol{b}$ means that

$$\boldsymbol{x}\wedge\boldsymbol{a}\wedge\boldsymbol{b} = 0 \Leftrightarrow \boldsymbol{x} = \alpha\boldsymbol{a} + \beta\boldsymbol{b}, \alpha, \beta \in \mathbb{R}. \tag{23}$$

Therefore a plane $\mathsf{B}$ is given by a simple bivector (also called 2-blade) $B = \boldsymbol{a}\wedge\boldsymbol{b}$ as

$$\mathsf{B} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}\wedge B = 0\}. \tag{24}$$

A three-dimensional volume subspace $\mathsf{C}$ is similarly given by a 3-blade $C = \boldsymbol{a}\wedge\boldsymbol{b}\wedge\boldsymbol{c}$ as

$$\mathsf{C} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}\wedge C = 0\}. \tag{25}$$

Finally a blade $D_r = \boldsymbol{b}_1\wedge\boldsymbol{b}_2\wedge\ldots\wedge\boldsymbol{b}_r, \boldsymbol{b}_l \in \mathbb{R}^n, 1 \le l \le r \le n$ describes an $r$-dimensional vector *subspace*

$$\mathsf{D} = \{\boldsymbol{x} \in \mathbb{R}^{p,q} | \boldsymbol{x}\wedge D = 0\}. \tag{26}$$

Its dual blade

$$D^* = Di_n^{-1} \tag{27}$$

describes the *complimentary* $(n-r)$-dimensional vector subspace $\mathsf{D}^{\perp}$. The magnitude of the blade $D_r \in Cl_n$ is nothing but the volume of the $r$-dimensional parallelepiped spanned by the vectors $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_r\}$.

Just as we were able to orthogonally reshaped a bivector to rectangular or square shape we can reshape every $r$-blade $A_r$ to a geometric product of mutually orthogonal vectors

$$A_r = \boldsymbol{a}'_1 \wedge \boldsymbol{a}'_2 \wedge \ldots \boldsymbol{a}'_r = \boldsymbol{a}_1 \boldsymbol{a}_2 \ldots \boldsymbol{a}_r, \tag{28}$$

with pairwise orthogonal and anticommuting vectors $\boldsymbol{a}_1 \perp \boldsymbol{a}_2 \perp \ldots \perp \boldsymbol{a}_r$. The reverse $\widetilde{A}_r$ of the geometric product of orthogonal vectors $A_r = \boldsymbol{a}_1 \boldsymbol{a}_2 \ldots \boldsymbol{a}_r$ is therefore clearly

$$(\boldsymbol{a}_1 \boldsymbol{a}_2 \ldots \boldsymbol{a}_r)^{\sim} = \boldsymbol{a}_r \ldots \boldsymbol{a}_2 \boldsymbol{a}_1 = (-1)^{\frac{r(r-1)}{2}} \boldsymbol{a}_1 \boldsymbol{a}_2 \ldots \boldsymbol{a}_r, \tag{29}$$

by simply counting the number $\frac{r(r-1)}{2}$ of permutations necessary.

Paying attention to the dimensions we find that the outer product of an $r$-blade $B$ with a vector $\boldsymbol{a}$ increases the dimension (grade) by $+1$

$$\boldsymbol{a} \wedge B = \langle \boldsymbol{a} B \rangle_{r+1}. \tag{30}$$

Opposite to that, the inner product (or left contraction) with a vector lowers the dimension (grade) by $-1$

$$\boldsymbol{a} \cdot B = \langle \boldsymbol{a} B \rangle_{r-1}. \tag{31}$$

The geometric product of two $r$-blades $A, B$ contains therefore at most the following grades

$$AB = \langle AB \rangle_0 + \langle AB \rangle_2 + \ldots + \langle AB \rangle_{2\min(r,[n/2])}, \tag{32}$$

where the limit $[n/2]$ (entire part of $n/2$) is due to the dimension limit of $\mathbb{R}^n$.

The inner product of vectors is properly generalized in geometric algebra by introducing the (left) contraction of the $r$-blade $A = A_r$ onto the $s$-blade $B = B_s$ as

$$A_r \rfloor B_s = \langle AB \rangle_{s-r}. \tag{33}$$

For blades of equal grade ($r = s$) we thus get the symmetric scalar

$$A_r \rfloor B_r = \langle AB \rangle_0 = \langle BA \rangle_0 = A * B. \tag{34}$$

Finally the product of a blade with its own reverse is necessarily scalar. Introducing orthogonal reshaping this scalar is seen to be

$$A_r \widetilde{A}_r = \boldsymbol{a}_1 \ldots \boldsymbol{a}_r \boldsymbol{a}_r \ldots \boldsymbol{a}_1 = \boldsymbol{a}_1^2 \ldots \boldsymbol{a}_r^2 = |A_r|^2, \tag{35}$$

therefore

$$|A_r| = |\boldsymbol{a}_1| \ldots |\boldsymbol{a}_r|. \tag{36}$$

Every $r$-blade $A_r$ can therefore be written as a product of the scalar magnitude $|A_r|$ times the geometric product of exactly $r$ mutually orthogonal unit vectors $\{\widehat{\boldsymbol{a}}_1, \ldots, \widehat{\boldsymbol{a}}_r\}$

$$A_r = |A_r| \widehat{\boldsymbol{a}}_1 \widehat{\boldsymbol{a}}_2 \ldots \widehat{\boldsymbol{a}}_r. \tag{37}$$

Please note well, that this *rewriting* of an $r$-blade in geometric algebra does not influence the overall result on the left side, the $r$-blade $A_r$ is before and after the rewriting the very same element of the geometric algebra $Cl_n$. But for the geometric interpretation of the geometric product $AB$ of two $r$-blades $A, B \in Cl_n$ the orthogonal reshaping is indeed a key step.

After a short discussion of reflections and rotations implemented in geometric algebra, we return to the geometric product of two $r$-blades $A, B \in Cl_n$ and present our key insight.

### 6.2 Reflections and rotations

A simple application of the geometric product is shown in Fig. 4 (left) to the reflection of a point vector $\boldsymbol{x}$ at a plane with normal vector $\boldsymbol{a}$, which means to reverse the component of $\boldsymbol{x}$ parallel to $\boldsymbol{a}$ (perpendicular to the plane)

$$\boldsymbol{x} \longrightarrow \boldsymbol{x}' = -\boldsymbol{a}^{-1} \boldsymbol{x} \boldsymbol{a}, \boldsymbol{a}^{-1} = \frac{\boldsymbol{a}}{\boldsymbol{a}^2}. \tag{38}$$

Two reflections lead to a rotation by twice the angle between the reflection planes as shown in Fig. 4 (right)

$$\boldsymbol{x} \longrightarrow \boldsymbol{x}'' = \boldsymbol{a}^{-1} \boldsymbol{b}^{-1} \boldsymbol{x} \boldsymbol{a} \boldsymbol{b} = (\boldsymbol{a}\boldsymbol{b}) \boldsymbol{x} \boldsymbol{a} \boldsymbol{b} = R^{-1} \boldsymbol{x} R, \tag{39}$$

with rotation operator (rotor) $R = \boldsymbol{a}\boldsymbol{b} \propto \cos\theta_{\boldsymbol{a},\boldsymbol{b}} + \mathbf{i}_{\boldsymbol{a},\boldsymbol{b}} \sin\theta_{\boldsymbol{a},\boldsymbol{b}}$, where the unit bivector $\mathbf{i}_{\boldsymbol{a},\boldsymbol{b}}$ represents the plane of rotation.

## 7 Geometric information in the geometric product of two subspace $r$-blades

From the foregoing discussion of the representation of $r$-dimensional subspaces $\mathsf{A}, \mathsf{B} \subset \mathbb{R}^n$ by the blades $A = \boldsymbol{a}'_1 \wedge \ldots \boldsymbol{a}'_r$ and $B = \boldsymbol{b}'_1 \wedge \ldots \boldsymbol{b}'_r$, from the freedom of orthogonally reshaping these blades and factoring out the blade magnitudes $|A|$ and $|B|$, and

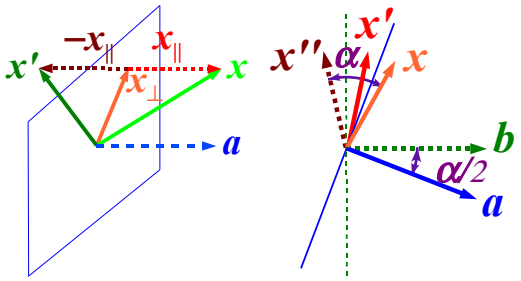



Figure 4: Reflection at a plane with normal $\boldsymbol{a}$ (left) and rotation as double reflection at planes normal to $\boldsymbol{a}$, $\boldsymbol{b}$ (right).

from the classical results of matrix algebra, we now know that we can rewrite the geometric product $AB$ in mutually orthogonal products of pairs of principal vectors $\boldsymbol{a}_k, \boldsymbol{b}_k, 1 \leq k \leq r$

$$A\widetilde{B} = \boldsymbol{a}_1\boldsymbol{a}_2\ldots\boldsymbol{a}_r\boldsymbol{b}_r\ldots\boldsymbol{b}_2\boldsymbol{b}_1 = \boldsymbol{a}_1\boldsymbol{b}_1\boldsymbol{a}_2\boldsymbol{b}_2\ldots\boldsymbol{a}_r\boldsymbol{b}_r. \tag{40}$$

The geometric product

$$\begin{aligned}\boldsymbol{a}_r\boldsymbol{b}_r &= |\boldsymbol{a}_r||\boldsymbol{b}_r|(\cos\theta_{\boldsymbol{a}_r,\boldsymbol{b}_r} + \mathbf{i}_{\boldsymbol{a}_r,\boldsymbol{b}_r}\sin\theta_{\boldsymbol{a}_r,\boldsymbol{b}_r}) \\ &= |\boldsymbol{a}_r||\boldsymbol{b}_r|(c_r + \mathbf{i}_r s_r),\end{aligned} \tag{41}$$

with $c_r = \cos\theta_{\boldsymbol{a}_r,\boldsymbol{b}_r}$ and $s_r = \sin\theta_{\boldsymbol{a}_r,\boldsymbol{b}_r}$ in the above expression for $A\widetilde{B}$ is composed of a scalar and a bivector part. The latter is proportional to the unit bivector $\mathbf{i}_r$ representing a (principal) plane orthogonal to *all* other principal vectors. $\mathbf{i}_r$ therefore commutes with all other principal vectors, and hence the whole product $\boldsymbol{a}_r\boldsymbol{b}_r$ (a rotor) commutes with all other principal vectors. A completely analogous consideration applies to all products of pairs of principal vectors, which proofs the second equality in (40).

We thus find that we can always rewrite the product $A\widetilde{B}$ as a product of rotors

$$\begin{aligned}&A\widetilde{B} \\ &= |A||B|(c_1 + \mathbf{i}_1 s_1)(c_2 + \mathbf{i}_2 s_2)\ldots(c_r + \mathbf{i}_r s_r) \\ &= |A||B|(c_1c_2\ldots c_r + \\ &\quad + s_1c_2\ldots c_r\mathbf{i}_1 + c_1s_2\ldots c_r\mathbf{i}_2 + \ldots + c_1c_2\ldots s_r\mathbf{i}_r + \\ &\quad\vdots \\ &\quad + s_1s_2\ldots s_r\mathbf{i}_1\mathbf{i}_2\ldots\mathbf{i}_r)\end{aligned} \tag{42}$$

We realize how the scalar part $\langle A\widetilde{B}\rangle_0 = |A||B|c_1c_2\ldots c_r$, the bivector part $\langle A\widetilde{B}\rangle_2 = |A||B|(s_1c_2\ldots c_r\mathbf{i}_1 + c_1s_2\ldots c_r\mathbf{i}_2 + \ldots + c_1c_2\ldots s_r\mathbf{i}_r)$, etc., up to the $2r$-vector (or $2[n/2]$-vector) part $\langle A\widetilde{B}\rangle_{2r} = |A||B|s_1s_2\ldots s_r\mathbf{i}_1\mathbf{i}_2\ldots\mathbf{i}_r$ of the geometric product $A\widetilde{B}$ arise and what information they carry.

Obviously the scalar part yields the cosine of the angle between the subspaces represented by the two $r$-vectors $A, B \in Cl_n$

$$\cos\theta_{\mathsf{AB}} = \frac{\langle A\widetilde{B}\rangle_0}{|A||B|} = \frac{A * \widetilde{B}}{|A||B|}, \tag{43}$$

which exactly corresponds to P.A. Wedin's formula from inner-product Grassmann algebra.

The bivector part consists of a sum of (principal) plane bivectors, which can in general be uniquely decomposed into its constituent sum of 2-blades by the method of Riesz, described also in [6], chapter 3-4, equation (4.11a) and following.

The magnitude of the $2r$-vector part allows to compute the product of all sines of principal angles

$$s_1s_2\ldots s_r = \pm\frac{|\langle A\widetilde{B}\rangle_{2r}|}{|A||B|}. \tag{44}$$

Let us finally refine our considerations to two general $r$-dimensional subspaces $\mathsf{A}, \mathsf{B}$, which we take to partly intersect and to be partly perpendicular. We mean by that, that the dimension of the intersecting subspace be $s \leq r$ ($s$ is therefore the number of principal angles equal zero), and the number of principle angles with value $\pi/2$ be $t \leq r - s$. For simplicity we work with normed blades (i.e. after dividing with $|A||B|$. The geometric product of the the $r$-blades $A, B \in Cl_n$ then takes the form

$$\begin{aligned}&A\widetilde{B} \\ &= (c_{s+1}c_{s+2}\ldots c_{r-t} + \\ &\quad + s_{s+1}c_{s+2}\ldots c_{r-t}\mathbf{i}_{s+1} + c_{s+1}s_{s+2}\ldots c_{r-t}\mathbf{i}_{s+2} + \\ &\quad \ldots + c_{s+1}c_{s+2}\ldots s_{r-t}\mathbf{i}_{r-t} + \\ &\quad\vdots \\ &\quad + s_{s+1}s_{s+2}\ldots s_{r-t}\mathbf{i}_{s+1}\mathbf{i}_{s+2}\ldots\mathbf{i}_{r-t})\mathbf{i}_{r-t+1}\ldots\mathbf{i}_r.\end{aligned} \tag{45}$$

We thus see, that apart from the integer dimensions $s$ for parallelity (identical to the dimension of the meet of blade $A$ with blade $B$) and $t$ for perpendicularity, the lowest non-zero grade of dimension $2t$ gives the relevant angular measure

$$\cos\theta_{\mathsf{AB}} = \cos\theta_{s+1}\cos\theta_{s+2}\ldots\cos\theta_{r-t}. \tag{46}$$

While the maximum grade part gives again the product of the corresponding sinuses

$$\sin\theta_{s+1}\sin\theta_{s+2}\ldots\sin\theta_{r-t}. \tag{47}$$

Dividing the product $A\widetilde{B}$ by its lowest grade part $c_{s+1}c_{s+2}\ldots c_{r-t}\mathbf{i}_{r-t+1}\ldots\mathbf{i}_r$ gives a multivector with maximum grade $2(r-t-s)$, scalar part one, and bivector part

$$t_{s+1}\mathbf{i}_{s+1}+t_{s+2}\mathbf{i}_{s+2}+\ldots+t_{r-t}\mathbf{i}_{r-t}, \tag{48}$$

where $t_k=\tan\theta_k$. Splitting this bivector into its constituent bivector parts further yields the (principal) plane bivectors and the tangens values of the principle angles $\theta_k, s<k\leq r-t$. This is the only somewhat time intensive step.

# 8 Conclusion

Let us conclude by discussing possible future applications of these results. The complete relative orientation information in $A\widetilde{B}$ should be ideal for a subspace structure self organizing map (SOM) type of neural network. Not only data points, but the topology of whole data subspace structures can then be faithfully mapped to lower dimensions. Our discussion gives meaningful results for partly intersecting and partly perpendicular subspaces. Apart from extracting the bivector components, all computations are done by multiplication. Projects like fast Clifford algebra hardware developed at the TU Darmstadt (D. Hildenbrand et al) should be of interest for applying the results of the paper to high dimensional data sets. An extension to offset subspaces (of projective geometry) and $r$-spheres (of conformal geometric algebra) may be possible.

# Acknowledgments

Soli deo gloria. I do thank my dear family, H. Ishi, D. Hildenbrand and V. Skala.

# References

[1] L. Andersson, The concepts of angle between subspaces ... unpublished notes, Umea (1980).

[2] P. A. Wedin, On angles between subspaces of a finite-dimensional inner product space, in Matrix Pencils, Bo Kagstram and Axel Ruhe, eds., Springer-Verlag, Berlin, 1983, pp. 263–285.

[3] L. Dorst et al, Geometric Algebra for Comp. Sc., Morgan Kaufmann, 2007.

[4] H. Li, Invariant Algebras and Geometric Reasoning, World Scientific, Singapore, 2008.

[5] Q.K. Lu, The elliptic geometry of extended spaces, Acta Math. Sinica, 13 (1963), pp. 49–62; translated as Chinese Math. 4 (1963), pp. 54–69.

[6] D. Hestenes, G. Sobczyk, Clifford Algebra to Geometric Calculus, Kluwer, 1984.

[7] E. Hitzer, K. Tachibana, S. Buchholz, I. Yu Carrier method for the general evaluation and control of pose, molecular conformation, tracking, and the like Advances in Applied Clifford Algebras, Vol. 19(2), pp. 339-364 (2009).

# Gaalop Compiler Driver

Patrick Charrier
TU Darmstadt, Germany
patrick.charrier@stud.tu-darmstadt.de

Dietmar Hildenbrand
TU Darmstadt, Germany
dhilden@gris.informatik.tu-darmstadt.de

**ABSTRACT**

The focus of the this work is on the better integration of algorithms expressed in Conformal Geometric Algebra (CGA) in modern high level computer languages, namely C++ and NVIDIA's Compute Unified Device Architecture (CUDA). A high runtime performance in terms of CGA is achieved using symbolic optimizing through the invocation of Gaalop.

**Keywords.** Conformal Geometric Algebra, Compiler Driver, Runtime Performance

## 1 INTRODUCTION

During the last decade Conformal Geometric Algebra (CGA) has become increasingly popular in expressing solutions to geometry related problems in scientific applications of robotics, dynamics, rendering and computer vision. Video game developers are becoming aware of CGA, in search for simpler and faster ways to describe their lighting [2] and physics algorithms. The majority of developers makes use of C-related programming languages like C++ or CUDA [14], which are performant and abstract enough for most needs.

From a programmer's perspective, the integration of CGA directly into C++ and CUDA yields a high level of intuitiveness. Coupled with a highly efficient generative software tool like Gaalop [9] in the background, an integration could set new standards to CGA-powered software development. The integration itself including other comforts, and to make CGA-usage available to a broad audience, is the purpose of this work.

### 1.1 Conformal Geometric Algebra

Conformal Geometric Algebra (CGA) is a new way of expressing most geometry focused mathematical problems. It deals naturally with intersections and transformations of planes, lines, spheres, circles, points and point pairs, but is also good at representing mechanics and dynamics. In Linear Algebra one would have to differentiate a plane-sphere intersection into three distinct cases, namely circle intersection, point intersection and no intersection. In Conformal Geometric Algebra the intersection itself is formulated as one operation on the plane (P) and the sphere (S) respectively.

$$R = S \wedge P$$

The three different cases of Linear Algebra are implicitly contained in the one result (R) of Conformal Geometric Algebra, being more compact and better readable. Similar observations can be made in other applications of geometry related mathematics. Applied to computer programs, CGA therefore has a high potential for improving code readability and to shorten production cycles. It has also been proven, that if implemented right, Geometric Algebra has at least similar performance, but sometimes even better performance, than conventional approaches [8].

An element of Conformal Geometric Algebra is referred to as multivector. A multivector consists of a linear combination of so called blades. Blades define the basis of CGA and are combinations of the vectors $e_1, e_2, e_3, e_0$ and $e_\infty$. All possible blades are listed in table 1. Keep this in mind for section 2.2.

| index | blade | grade |
|---|---|---|
| 1 | 1 | 0 |
| 2 | $e_1$ | 1 |
| 3 | $e_2$ | 1 |
| 4 | $e_3$ | 1 |
| 5 | $e_\infty$ | 1 |
| 6 | $e_0$ | 1 |
| 7 | $e_1 \wedge e_2$ | 2 |
| 8 | $e_1 \wedge e_3$ | 2 |
| 9 | $e_1 \wedge e_\infty$ | 2 |
| 10 | $e_1 \wedge e_0$ | 2 |
| 11 | $e_2 \wedge e_3$ | 2 |
| 12 | $e_2 \wedge e_\infty$ | 2 |
| 13 | $e_2 \wedge e_0$ | 2 |
| 14 | $e_3 \wedge e_\infty$ | 2 |
| 15 | $e_3 \wedge e_0$ | 2 |
| 16 | $e_\infty \wedge e_0$ | 2 |
| 17 | $e_1 \wedge e_2 \wedge e_3$ | 3 |
| 18 | $e_1 \wedge e_2 \wedge e_\infty$ | 3 |
| 19 | $e_1 \wedge e_2 \wedge e_0$ | 3 |
| 20 | $e_1 \wedge e_3 \wedge e_\infty$ | 3 |
| 21 | $e_1 \wedge e_3 \wedge e_0$ | 3 |
| 22 | $e_1 \wedge e_\infty \wedge e_0$ | 3 |
| 23 | $e_2 \wedge e_3 \wedge e_\infty$ | 3 |
| 24 | $e_2 \wedge e_3 \wedge e_0$ | 3 |
| 25 | $e_2 \wedge e_\infty \wedge e_0$ | 3 |
| 26 | $e_3 \wedge e_\infty \wedge e_0$ | 3 |
| 27 | $e_1 \wedge e_2 \wedge e_3 \wedge e_\infty$ | 4 |
| 28 | $e_1 \wedge e_2 \wedge e_3 \wedge e_0$ | 4 |
| 29 | $e_1 \wedge e_2 \wedge e_\infty \wedge e_0$ | 4 |
| 30 | $e_1 \wedge e_3 \wedge e_\infty \wedge e_0$ | 4 |
| 31 | $e_2 \wedge e_3 \wedge e_\infty \wedge e_0$ | 4 |
| 32 | $e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$ | 5 |

Table 1: The 32 blades of 5D Conformal Geometric Algebra, that compose a multivector.

### 1.2 High Level Programming Languages

Modern very high level software development tools, like Java [15], define a very abstract language, on which programmers and scientists can work in a natural way and with results of moderate performance. Machine level languages on the other hand, like Assembler, tend to produce very fast results, but with less intuition,

which often leads to longer and more costly development cycles.

In order to shorten development time and to produce fast code at the same time, the solution lies somewhere in between. Object oriented programming languages like C++, C#, Objective Pascal and Smalltalk provide a good level of abstraction, but also excellent performance. They seem to be a good choice for most modern scientific and business projects and are therefore the most common languages, leaded by C and C++. Recently NVIDIA's Compute Unified Architecture (CUDA) programming language enabled users to utilize the very high computing power of modern graphics chips. CUDA device code is a subset of the common C language with some extensions added to it.

We strongly believe, that the applications of Conformal Geometric Algebra are most likely to be found in high performance applications, such as games, industrial or scientific software. Since, as described above, C-like languages are leading the field of high performance computing, possible integration of CGA code into C/C++/CUDA has the most advantage.

## 2 RELATED WORK

Combining both the aspects of Conformal Geometric Algebra and Modern Programming Languages (namely C++ and CUDA), promises to have a high potential for scientific work. Unfortunately CGA has such a high level of abstraction, that it does not naturally fit into C++ and CUDA programs. In order to solve this problem and to make CGA run fast, recent approaches try to wrap CGA into templated multivector classes (Gaalet [18]) or make use of a domain specific language (DSL) as input language for a code generator (Gaigen2 [5], GMac [4] and Gaalop [9]). All software tools are very well suited in their domain and produce good results. In the following, we will present the CLUScript language, Gaalop and the Compiler Driver concept.

### 2.1 CLUScript

Conformal Geometric Algebra can not be expressed in terms of regular mathematical syntax. CGA-specific operators like the outer product $\wedge$, inner product . and geometric product $*$ require special treatment in regular programming languages or the definition a completely new domain specific language (DSL).

The DSL that powers this work is CLUScript. The especially designed integrated development environments for CLUScript are called CLUCalc (old) and CLUViz (new). In words of the author Dr. Christian Perwass [17, 16].

> CLUCalc/CLUViz is a freely (for non-commercial use) available software tool for 3D visualizations and scientific calculations that was conceived and written by Dr. Christian Perwass. CLUCalc interprets a script language called CLUScript, which has been designed to make mathematical calculations and visualizations very intuitive.

Indeed, CLUScript is a very intuitive language and we have found CLUCalc to be an advanced tool for developing and testing Geometric Algebra algorithms. It is easy to use, installs and runs smoothly on Microsoft Windows platforms. Unfortunately, the support for Linux or Macintosh platforms is very limited, but it may run with some effort.

### 2.2 Gaalop

The Geometric Algebra Algorithms Optimizer (Gaalop) [9] was developed by TU Darmstadt (Germany) and is a powerful tool for optimizing algorithms, expressed in Conformal Geometric Algebra. It generates non CGA-specific code from code defined in a CGA-specific language and symbolically optimizes the algorithm on-the-fly, invoking a Computer Algebra System (CAS). In this context, CGA can be seen as a higher level mathematical language that is being transformed into simple arithmetic mathematical language by Gaalop. Philosophically spoken, Gaalop could be defined as a math compiler.

CLUScript as an input language and C/C++ as output language has proven to be an extremely powerful combination. It is also possible to generate Field Programmable Array (FPGA), LaTex and CLUScript representations. For evaluation purposes it is often helpful to choose CLUScript output, then replace the original CLUScript code with the optimized code and test the result for the same functionality as the original code. Generated Gaalop C/C++ code has to be pasted into the final code and requires additional handwork.

### 2.3 Compiler Drivers and CUDA

The Compiler Driver concept is a very simple, but powerful approach to extend the features of a programming language. It has recently been used by NVIDIA's CUDA [14]. We will use CUDA as an example to explain the usage of compiler drivers. A traditional C++ program is separated into several source files. The source files will then be converted to an intermediate form by the C++ compiler. This intermediate form is called an object file. All the object files are then linked together by the linker, resulting in the final executable file. For instance, see the following example in figure 1.

The language syntax of CUDA is fully compatible with C++. One may compile any C++ code with NVIDIA's NVCC compiler without any modifications to the original code. The produced machine code, however, will still run on the host.

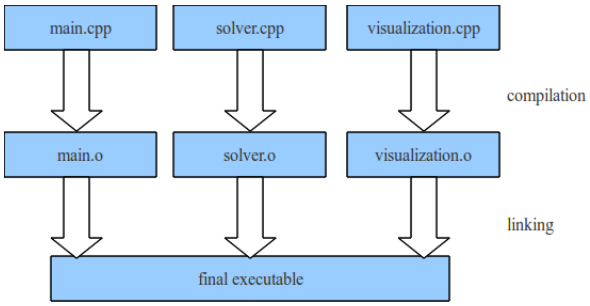



Figure 1: C++ compilation process.

In order to make use of the graphics chip's high number of streaming processors, we will have to include CUDA-specific language extensions like the *__global__* or *__device__* keywords and kernel call statements, which are not part of the original C++ language. The compiled object file will seamlessly link with the code compiled with the regular C++ compiler. To make this clear, see the simplified diagram in figure 2. The solver.cu source file is CUDA code, all other files are C++ code.

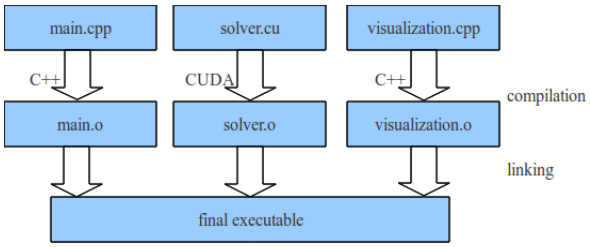



Figure 2: Simplified CUDA compilation process.

Notice how the CUDA compiler seamlessly integrates with the C++ compiler. With the *__global__* and *__device__* and other keywords NVIDIA has extended the C++ language syntax. But instead of creating their own C++ compiler from scratch, they came up with the approach of reusing the existing C++ compiler and linker. This is called the Compiler Driver concept.

From the NVIDIA CUDA Programming Guide 3.0 [3].

> Source files compiled with nvcc can include a mix of host code (i.e. code that executes on the host) and device code (i.e. code that executes on the device). nvcc's basic workflow consists in separating device code from host code and compiling the device code into an assembly form (PTX code) and/or binary form (cubin object). The generated host code is output either as C code that is left to be compiled using another tool or as object code directly by letting nvcc invoke the host compiler during the last compilation stage.

This way CUDA makes full usage of the existing C++ compiler and linker features, extends them, but separates both compilers at the same time. This enables lower complexity and better maintenance of NVCC. The full diagram can be seen in figure 3.

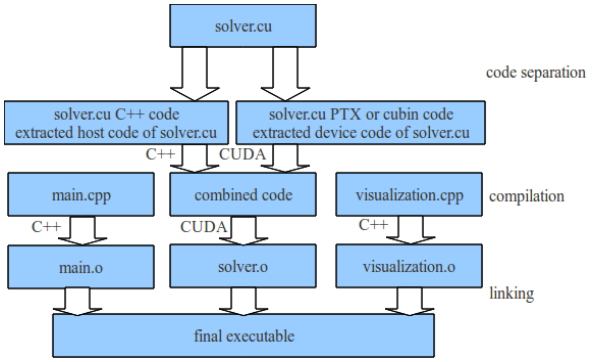



Figure 3: Full CUDA compilation process.

## 3 GAALOP COMPILER DRIVER

We have presented Conformal Geometric Algebra, CLUScript, Gaalop and C++. All these aspects are somehow related to each other, but simply not linked yet. This is where the Compiler Driver concept comes in and elegantly connects all parts. The diagram in figure 4 shows, how this will be achieved in particular.

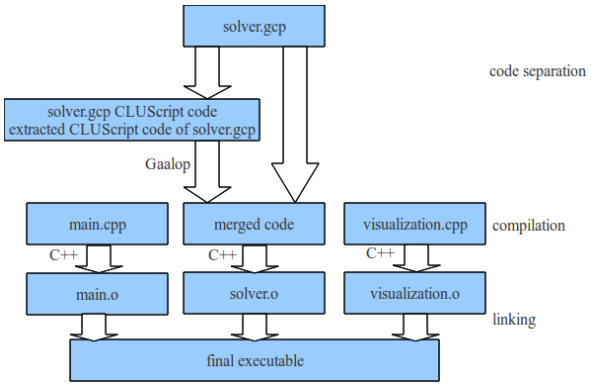



Figure 4: GCD C++ compilation process.

We now use the file extension .gcp for CLUScript-extended C++ source files. The particular steps, that occur when compiling .gcp-files, are the following.

1. The user issues the build command. This can happen in an integrated development environment (IDE) using custom build rules, as well as using *GNU make* [6] or other build automation tools.
2. The build tool passes .gcp-files to Gaalop Compiler driver (GCD) over the command line.
3. GCD extracts the CLUScript parts of the .gcp-file and writes them into separate files.
4. GCD invokes Gaalop over its new command line interface, passing the extracted code files, one at a time.

5. Gaalop symbolically simplifies the extracted code files.

6. GCD merges the returned code into with the original code, exactly where the *pragmas* are.

7. GCD invokes the regular C++ compiler passing the merged C++ file.

8. Finally, the C++ compiler produces an object file, which seamlessly integrates into the linking process.

### 3.1 Gaalop Compiler Driver for CUDA

Note that the concept is not restricted to C++. It can be applied to CUDA or other programming languages in the same manner. The resulting diagram in figure 5 for CUDA is slightly more complex, as GCD passes its data to NVCC, which itself is a compiler driver.

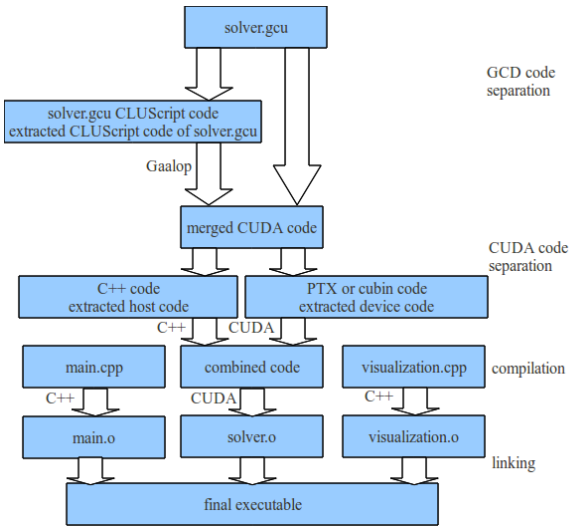



Figure 5: GCD CUDA compilation process.

We choose the file extension .gcu for CLUScript-extended CUDA programs. The compilation process steps remain the same as with C++ GCD, with the exception of passing the generated file to NVCC instead of the C++ compiler in steps 7 and 8.

## 4 A GUIDE TO GAALOP GCD

The following section shows, how to make use of GCD in real world-applications. It is intended as a quick start-guide, described by three example code snippets.

### 4.1 The Test Case

The code in the following listings was extracted from a Molecular Dynamics Simulation currently under development by us and the High Performance Computing Center Stuttgart (HLRS) and is partially optimized with Gaalop and GCD. A molecular dynamics simulation models the point-pair interactions of a system of molecules, each one consisting of several atoms, and numerically solves Newton's and Euler's equations of motion for each molecule.

The aim of the project is a runtime comparison between a conventional solver and several implementations of a new formalism based on Hestenes' work on screw mechanics described in CGA [7], including a compact formulation of combined translational and rotational dynamics within a velocity verlet algorithm. Details of the Molecular Dynamics formalism used as a test case in the present work is intended to be part of a future publication. The CGA solvers were implemented with Gaalet, Gaalop and GCD running on the CPU, as well as Gaalop and GCD running on CUDA.The listings show code extracted from both the CPU and CUDA versions of the Gaalop and GCD solvers.

### 4.2 Code examples for C++

The whole simulation was firstly implemented in CLUCalc using the CLUScript language and later ported to C++ using Gaalop. Listing 1 shows the initialization of a particular molecule, taken from the original CLUScript. Location and orientation are defined through the molecule's versor *D_result* (refer to [7]). Linear and angular velocity are defined through the molecule's velocity screw *V_result*. The consecutive simulation steps start with the values computed in this initialization code.

```
// create versor from input values
rotor = arw + arx*e2^e3 + ary * e3^e1 + arz * e1^e2;
translator = 1 - 0.5*(lpx*e1 + lpy*e2 + lpz*e3)^einf;
?D_result = translator*rotor;

// create velocity screw from input values
lv = lvx*e1+lvy*e2+lvz*e3;
av = avx*e1+avy*e2+avz*e3;
?V_result = einf*lv - e1^e2^e3*av;
```

**Listing 1: Original CLUScript input for Gaalop.**

Note that *D_result* and *V_result* are already declared for export in Gaalop indicated by the question marks. The resulting Gaalop output code is then directly pasted into the target C++ file, as can be seen in listing 2.

```
// map molecule data to gaalop data
const float lpx = molecule.lpos[0];
const float lpy = molecule.lpos[1];
const float lpz = molecule.lpos[2];
const float arw = molecule.arot[0];
const float arx = -molecule.arot[1];
const float ary = -molecule.arot[2];
const float arz = -molecule.arot[3];
const float lvx = molecule.lvel[0];
const float lvy = molecule.lvel[1];
const float lvz = molecule.lvel[2];
const float avx = molecule.avel[0];
const float avy = molecule.avel[1];
const float avz = molecule.avel[2];

// gaalop generated code
float D_result_opt[32] = {0.0f};
D_result_opt[1]=arw;
D_result_opt[7]=arz;
D_result_opt[8]=-ary;
D_result_opt[9]=0.5*lpy*arz-0.5*lpx*arw-0.5*lpz*ary;
D_result_opt[11]=arx;
D_result_opt[12]=0.5*lpz*arx-0.5*lpy*arw-0.5*lpx*arz;
D_result_opt[14]=-0.5*lpz*arw+0.5*lpx*ary-0.5*lpy*arx;
D_result_opt[27]=-0.5*lpy*ary-0.5*lpz*arz-0.5*lpx*arx;
```

```
// gaalop generated code
float V_result_opt[32] = {0.0f};
V_result_opt[7]=-avz;
V_result_opt[8]=avy;
V_result_opt[9]=-lvx;
V_result_opt[11]=-avx;
V_result_opt[12]=-lvy;
V_result_opt[14]=-lvz;

// map gaalop data to molecule data
GaalopMapVersor(D, D_result_opt);
GaalopMapVelocityScrew(V, V_result_opt);
```

**Listing 2: Merged Gaalop and C++ code.**

Notice that the result code not only contains the generated Gaalop code, but also several variable and array assignments. Those assignments are data mappings between the original molecule data structure and the generated Gaalop code. They are quite common for most Gaalop-powered applications.

The function *GaalopMapVersor* assigns the elements 1,7,8,9,11,12,14,27 of array *D_result_opt* to the elements 0,1,2,3,4,5,6,7 of array *D*, that was previously declared with size 8. The function *GaalopMapVelocityScrew* assigns the elements 7,8,9,11,12,14 of array *V_result_opt* to the elements 0,1,2,3,4,5 of array *V*, that was previously declared with size 6. The code in between the two mapping blocks is the actual Gaalop output code.

One may modify the array indices and variable names by hand to improve speed and to avoid data mappings. For reasons of transparency this is usually not a good choice, meaning that if we might discover a bug or we would like to include a new feature in one of our CLUScript files, we again have to modify the generated code by hand. As there might be a large number of Gaalop-generated code snippets in a GGA-powered C++ program, this process will increase the probability of bugs and development time. Using data-mappings enables us to re-paste modified code at any time.

Also notice that even without the data mappings, the generated code is hardly interpretable by human means. Keeping this in mind, review the following code.

```
  // map molecule data to gaalop data
  ...

#pragma gcd begin
  // create versor from input values
  rotor = arw + arx*e2^e3 + ary * e3^e1 + arz * e1^e2;
  translator = 1 - 0.5*(lpx*e1 + lpy*e2 + lpz*e3)^einf;
  ?D_result = translator*rotor;

  // create velocity screw from input values
  lv = lvx*e1+lvy*e2+lvz*e3;
  av = avx*e1+avy*e2+avz*e3;
  ?V_result = einf*lv - e1^e2^e3*av;
#pragma gcd end

  // map gaalop data to molecule data
  GaalopMapVersor(D, D_result);
  GaalopMapVelocityScrew(V, V_result);
```

**Listing 3: Gaalop Compiler Driver for C++ input code.**

**Note:** The preceding variable mappings were removed in order to keep the size of this document small. Please consider them to be in place when evaluating the code.

The CLUScript code is now directly embedded in the C++ code in between the *gcd pragmas*, instead of the pasted Gaalop code. As result, this reduces the source code size and makes it much better readable.

Since the code now contains CLUScript statements, which are apparently not part of the C++ standard, we will not be able to compile it with a regular C++ compiler. To be specific, the C++ standard is being extended using the Compiler Driver concept, as stated in section 3.

## 4.3 Code example for CUDA

The example above was chosen, because it includes a lot of CGA-statements and is easy to understand. It is also possible to include this code into a CUDA-Kernel, but not meaningful here. The code shown is only called once for each molecule before the simulation, and never called again. Wisely chosen CUDA-Kernels are called one or many times per frame, as the one in listing 4. Again, it is taken from the original simulation, with some parts removed.

```
__device__ void addMoleculeForceAndTorque(
        float* mol_lmom,
        float* mol_amom,
        const float* versor1,
        const float3& localPos,
        const float3& globalForce,
        ...)
{
// map molecule data to gaalop data
const float Di = versor1[0];
const float D12 = versor1[1];
const float D13 = versor1[2];
const float D1x = versor1[3];
const float D23 = versor1[4];
const float D2x = versor1[5];
const float D3x = versor1[6];
const float D123x = versor1[7];

const float px = localPos.x;
const float py = localPos.y;
const float pz = localPos.z;

const float fgx = globalForce.x;
const float fgy = globalForce.y;
const float fgz = globalForce.z;

#pragma gcd begin
  // input values
  moleculeVersor = Di + D23*e2^e3 + D13*e1^e3
                      + D12*e1^e2 + D1x*e1^einf
                      + D2x*e2^einf + D3x*e3^einf
                      + D123x*e1^e2^e3^einf;
  posLocal = px*e1 + py*e2 + pz*e3;
  forceGlobal = fgx*e1 + fgy*e2 + fgz*e3;

  // final values
  ?result_force = ~moleculeVersor
                      * forceGlobal
                      * moleculeVersor;
  ?result_torque = posLocal ^ result_force;
#pragma gcd end

// add resulting force and torque to molecule's data
...
}
```

**Listing 4: Gaalop Compiler Driver for CUDA input code.**

# 5 RESULTS

## 5.1 Performance and Compile Time

Figure 6 shows the performance of the GCD solver versus the Conventional solver for the Test Case in section 4.1 on a quadcore machine. The GCD solver is slightly faster or equally fast compared to the Conventional solver, which is not self-evident for CGA-based implementations of such complexity. Additionally, the GCD solver has a more compact code, as described in section 4.1. Notice, that section 6 shows ways to achieve more than two times faster results, by removing unused memory and caching artifacts (figure 7).

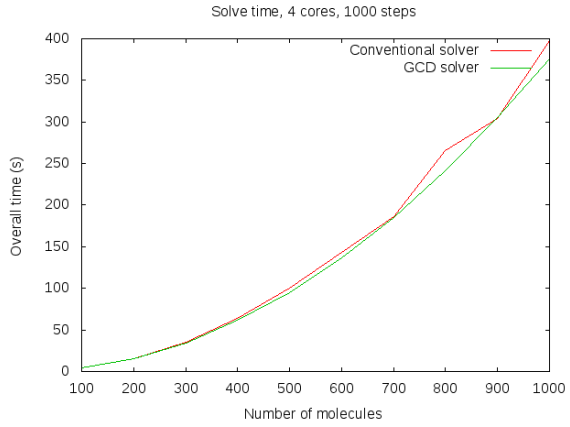



Figure 6: Performance Results - GCD solver is slightly faster or equally fast compared to Conventional solver. Much higher Performance is achieved in figure 7, Future Work section 6.

Compilation time is 1.920s for the Conventional versus 10.426s for the GCD solver. The prolonged compile time is due to the Gaalop-performed symbolic optimizations in the background.

## 5.2 CMake Support

Development with most programming languages, especially C++, is highly dependent on specifying build logic. Build logic explicitly defines which source files need to be compiled with which tool, and how the resulting intermediate files get linked together into the final executable or library file. Integrated development environments (IDE) like *Microsoft Visual Studio* [13] or *Code::Blocks* [1] automatically manage the default parts of the build logic.

However, with a rising number of operating systems, compilers and build tools, it has become very difficult to maintain the build logic for every possible combination of operating system and compiler. *CMake* [12] elegantly solves this problem by acting as a build logic generator. More detailed, *CMake* defines a script language, that is independent of the build platform. This script language, is then transformed into the target platform definition, e.g. **.sln* project files for *Visual Studio* or *Makefiles* for *GNU make* [6].

*CMake* is rapidly becoming the de facto standard for cross platform build tools. It also supports automatic unit testing (*CTest*), install and deploy mechanisms (*CPack*), and web-based error reporting (*CDash*).

*CMake* support for GCD is provided by a CMake-script named *FindGCD.cmake*. If the script is installed in CMake's *Modules* subdirectory, it can be invoked by CMake's *FIND_PACKAGE(GCD)* command. Libraries and executables containing GCD code may be built using *GCD_CPP_ADD_LIBRARY* and *GCD_CPP_ADD_EXECUTABLE* commands. GCD for CUDA builds use *GCD_CUDA_ADD_LIBRARY* and *GCD_CUDA_ADD_EXECUTABLE*.

An example *CMakeLists.txt* build script is shown in listing 5.

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
FIND_PACKAGE(GCD)
GCD_CPP_ADD_EXECUTABLE(test1 "Test1_PointTriangle.gcp")
ADD_TEST(NAME "Test1_PointTriangle" COMMAND test1)
```

**Listing 5: Example CMake build script using GCD.**

Given this definition, *CMake* compiles and links the GCD source file *"Test1_PointTriangle.gcp"* into an application *test1*, and specifies it as a runtime test. *CTest* runs the executable and reports it as PASSED, if its return value is zero, or FAILED, if it is non-zero. Tests like this one are an important part of software quality assurance.

## 5.3 GCD helper library

The goal behind GCD helper library is to assist users of GCD by providing essential functions, that simplify development of CGA-powered applications. It is intended as a multi-purpose library, adaptable to a broad range of applications working with Conformal Geometric Algebra. Reoccurring tasks, like the mapping of versors (*GaalopMapVersor* from section 4.2), are implemented within the library. It also provides C++-macros, that state the position of a particular multivector entry, e.g. $e1 \wedge e2$ is listed as *"#define E12 6"*. For example, *D[E12]* returns the blade $e1 \wedge e2$ of a multivector $D$ (see table 1 for a refresh of the 32 blades of CGA).

The GCD helper library is automatically linked when using *CMake*, as described in section 5.2. However, the library's header files have to be included with *"#include <gcd.h>"* in **.gcp* (GCD for C++) or **.gcu* (GCD for CUDA) source files.

# 6 FUTURE WORK

The presented approach still has a lot of potential to improve on. Further work has to be put into simplifying data mappings and advancing memory usage. Gaalop automatically allocates arrays of 32 floating point numbers to make space for all possible multivector entries, which results in a memory usage of 128 bytes per multivector (see subsection 1.1). Most multivectors usually

contain about up to 8 entries, which results in about 24 unused multivector entries and 96 bytes of unneeded memory. It is not trivial to reduce the required space, because we must deal with the theoretical assumption, that all multivector entries could be assigned.

Using data mappings, the effect can be hidden, and no useless data will be saved and read from RAM, but it still occupies register space and cache, which has an effect on performance. This turned out to to be a major bottleneck in our Molecular Dynamics simulation.

Listing 6 shows an outlook of how future GCD code may look like.

```
// map molecule data to gaalop data
...

#pragma gcd begin
  // create versor from input values
  rotor = arw + arx*e2^e3 + ary * e3^e1 + arz * e1^e2;
  translator = 1 - 0.5*(lpx*e1 + lpy*e2 + lpz*e3)^einf;
  ?D = translator*rotor;

  // create velocity screw from input values
  lv = lvx*e1+lvy*e2+lvz*e3;
  av = avx*e1+avy*e2+avz*e3;
  ?V = einf*lv - e1^e2^e3*av;
#pragma gcd end
```

**Listing 6: Future GCD code without subsequent data mappings.**

Notice that *D* and *V* do not need to be saved into temporary arrays *D_result* and *V_result* anymore. They are directly stored into the final arrays, saving additional copy time, register and cache usage. Preliminary tests on this subject reduced the runtime of our test case down to 36 percent of its original value and are very promising (figure 7).

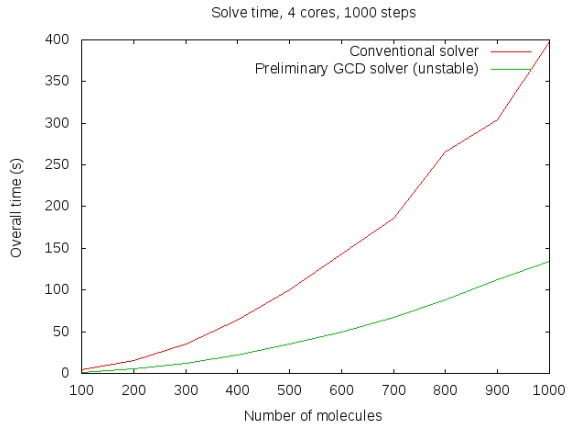



Figure 7: Future Performance Results

Ongoing work is being put into symbolically optimizing larger parts of the CLUScript-syntax with Gaalop. For example, *while-loops* can be unrolled and symbolically treated in the same way as other code.

Apart from C++ and CUDA, other languages like OpenCL [11], Microsoft DirectCompute and shading languages (CG, HLSL) are interesting target languages for GCD and promising topics for further research.

The GCD helper library will be expanded in future work to support direct rendering of multivectors similar to CLUCalc (see figure 8 for example). That is, given a particular multivector *m*, the helper library will firstly determine its representation in three-dimensional space (e.g. sphere, plane, circle, line, point-pair or point). Given the representation and its parameters, the library will render the appropriate object with OpenGL [10] or other rendering APIs.

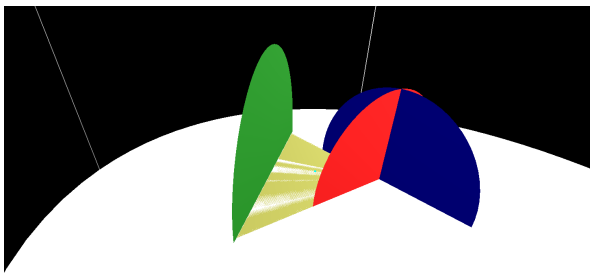


Figure 8: An example of CLUCalc generated graphics

## 7 CONCLUSION

Code simplicity, elegance and intuitiveness are the major goals of this work. Recalling the code examples shows that these goals were reached. As GCD directly profits from any improvements within Gaalop through its invocation, a high runtime performance is achieved on-the-fly.

Gaalop GCD symbolically optimizes the embedded CLUScript code in order to improve runtime. A longer compile time is a natural consequence of the concept. However, we do not recommend putting much research into this aspect, as the build process can already be parallelized in many build automation tools like *GNU make* [6]. We found, that in reality, using parallelized builds, longer compile time is not a problem.

We would like to conclude, that Gaalop Compiler Driver has the potential to change the way programmers work with Conformal Geometric Algebra inclusions in their code. Instead of separating code generation and code compilation into two distinct processes, it is now a single simplified process. Especially the combination of CGA and CUDA enables new methods for research. As it is now easier to develop with, we hope that more scientists, game and software programmers will find their way into the applications of Conformal Geometric Algebra.

## REFERENCES

[1] Code::blocks. http://www.codeblocks.org.

[2] The homepage of geomerics ltd. Available at `http://www.geomerics.com`.

[3] NVIDIA Corporation. *NVIDIA CUDA Programming Guide 3.0*, 2010. Available at `www.nvidia.com`.

[4] Ahmad Hosney Awad Eid. *Optimized Automatic Code Generation for Geometric Algebra Based Algorithms with Ray Tracing Application*. PhD thesis, Port-Said, 2010.

[5] Daniel Fontijne, Tim Bouma, and Leo Dorst. Gaigen 2: A geometric algebra implementation generator. Available at `http://staff.science.uva.nl/~fontijne/gaigen2.html`, 2007.

[6] Free Software Foundation. Gnu make. http://www.gnu.org/software/make.

[7] David Hestenes. Old wine in new bottles : A new algebraic framework for computational geometry. In Eduardo Bayro-Corrochano and Garret Sobczyk, editors, *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser, 2001.

[8] Dietmar Hildenbrand, Daniel Fontijne, Yusheng Wang, Marc Alexa, and Leo Dorst. Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In *Eurographics conference Vienna*, 2006.

[9] Dietmar Hildenbrand, Joachim Pitt, and Andreas Koch. *Gaalop - High Performance Parallel Computing based on Conformal Geometric Algebra*, volume 1, pages 350–358. Springer, May 2010.

[10] Khronos. *OpenGL Specifications*, 2010. http://www.opengl.org/documentation/specs/.

[11] Khronos-Group. The OpenCL home page. Available at `http://www.khronos.org/opencl/`, 2009.

[12] Inc. Kitware. Cmake cross platform make. http://www.cmake.org.

[13] Microsoft. Microsoft visual studio. http://www.microsoft.com/visualstudio.

[14] NVIDIA. The CUDA home page. Available at `http://www.nvidia.com/object/cuda\_home.html`, 2010.

[15] Oracle. Java. http://java.com/en/.

[16] Christian Perwass. *Geometric Algebra with Applications in Engineering*. Springer, 2009.

[17] Christian Perwass. The CLU home page. Available at http://www.clucalc.info, 2010.

[18] Florian Seybold. Geometric algebra algorithm expression templates. http://sourceforge.net/projects/gaalet/develop, 2010.

# Registration of Multichannel Images using Geometric Algebra

Andreas Görlitz
Darmstadt University of Technology
Department of Computer Science
Hochschulstr. 10
64289 Darmstadt, Germany
A.Goerlitz@stud.tu-darmstadt.de

Helmut Seibert
Fraunhofer Institute for
Computer Graphics Research
Fraunhoferstr. 5
64283 Darmstadt, Germany
helmut.seibert@igd.fraunhofer.de

Dietmar Hildenbrand
Darmstadt University of Technology
Department of Computer Science
Hochschulstr. 10
64289 Darmstadt, Germany
dhilden@gris.informatik.tu-darmstadt.de

## ABSTRACT

Geometric Algebra (GA) is a mathematical framework that allows a compact and geometrically intuitive description of geometric relationships and algorithms. In this paper a translation, rotation and scale invariant algorithm for registration of color images and other multichannel data is introduced. The use of Geometric Algebra allows to generalize the well known Fourier Transform which is widely used for the registration of scalar fields. In contrast to the original algorithm our algorithm allows to handle vector valued data in an appropriate way. As a proof of concept the registration results for artificial, as well as for real world data, are discussed.

## Keywords

Registration, Multichannel Images, Color Images, Clifford Fourier Transform, Geometric Algebra (GA)

## 1 INTRODUCTION

Registration of images is a crucial step in many image processing applications where the final information is obtained by combining multiple input images. Widely used applications such as image stitching [16], medical imaging [3] and video tracking [15], heavily rely on the accuracy of image registration. A broad variety of approaches for various image registration problems has been developed and presented in the literature, a survey which classifies the different approaches is given in [19].

In many applications multi-channel images are available, which require adequate processing of vector data. Fundamental image processing steps such as convolution and correlation are not well suited to work with vector data, as the multiplication of vectors has a different meaning as the multiplication of scalar values. In practice there are two common ways to work around this limitation. One is to reduce the dimensionality, e.g. to convert color images to a monochrome representation. The other way is to handle the vector components respective channels independently and to combine the results afterwards, e.g. perform a filter operation on the red, green, and blue channel of a RGB image separately.



Obviously both approaches cause inaccuracies by introducing information loss or misinterpretations and may lead to inappropriate results.

The key innovation of our work presented here is a way to perform a registration of multichannel images based on the Fourier Transform using Geometric Algebra as an adequate mathematic foundation to handle vector data in a well defined and appropriate way.

In Section 3 a brief introduction of some Geometric Algebra basics is given and the registration of images using the Fourier Transform is discussed. As main contribution of this paper the generalization of this registration approach for multichannel images, such as color images, introduced in Section 4. The evaluation and the results are discussed in Section 5 and 6 respectively, and afterwards a conclusion is given in Section 7. Finally an outlook to the future work is given in Section 8.

## 2 RELATED WORK

Applications that rely on registration of images are a.o.: image stitching [16], medical imaging [3] and video tracking [15]. Especially in medical imaging the registration of multichannel images plays an important role [12], [14]. To be able to process these multichannel images in the phase domain, a generalization of the classical Fourier-Transform is needed. Many different approaches have been invented to generalize the classical Fourier Transform in the recent years [2], [7] and [8]. The basis of this paper is the so called Clifford-Fourier-Transform that was derived and successfully used in [5] and [6].

# 3 PREREQUISITES

## 3.1 Geometric Algebra

Below we only describe the parts of the *Geometric Algebra* on $\mathbb{R}^n$, $\mathcal{G}(\mathbb{R}^n)$ or in abbreviate form $\mathcal{G}_n$, that will be used in this paper. For a more detailed introduction to Geometric Algebra we reference to [1], [4], [11], [18].

**Inner Product:** let $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathcal{G}_n$ be Geometric Algebra vectors and $k \in \mathbb{R}$, then the inner product $\mathbf{a}\cdot\mathbf{b}$, that is also known as the *scalar product* from vector calculus, has the following properties:

$$\begin{aligned}\mathbf{a}\cdot\mathbf{b} &= \mathbf{b}\cdot\mathbf{a}\\ (k\mathbf{a})\cdot\mathbf{b} &= k(\mathbf{a}\cdot\mathbf{b})\\ \mathbf{a}\cdot(\mathbf{b}+\mathbf{c}) &= \mathbf{a}\cdot\mathbf{b}+\mathbf{a}\cdot\mathbf{c}\ .\end{aligned}$$

Further we also know from vector calculus that $\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos(\alpha)$, where $\alpha$ is the angle between $\mathbf{a}$ and $\mathbf{b}$, i.e. $\mathbf{a}\cdot\mathbf{b} = 0 \;\; \mathbf{a},\mathbf{b} \neq 0 \Leftrightarrow \mathbf{a}$ and $\mathbf{b}$ are orthogonal.

**Outer Product:** let $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathcal{G}_n$ be Geometric Algebra vectors and $k \in \mathbb{R}$, then the outer product $\mathbf{a}\wedge\mathbf{b}$ has the following properties:

$$\begin{aligned}\mathbf{a}\wedge\mathbf{b} &= -\mathbf{b}\wedge\mathbf{a}\\ (\mathbf{a}\wedge\mathbf{b})\wedge\mathbf{c} &= \mathbf{a}\wedge(\mathbf{b}\wedge\mathbf{c})\\ \mathbf{a}\wedge(\mathbf{b}+\mathbf{c}) &= \mathbf{a}\wedge\mathbf{b}+\mathbf{a}\wedge\mathbf{c}\\ (k\mathbf{a})\wedge\mathbf{b} &= k(\mathbf{a}\wedge\mathbf{b})\ .\end{aligned}$$

It can be shown that the outer product $\mathbf{a}\wedge\mathbf{b}$ spans a plane, i.e. $\mathbf{a}\wedge\mathbf{b} = 0 \Leftrightarrow \mathbf{a}$ and $\mathbf{b}$ are parallel.

A term like $\bigwedge_{i=1}^{k} b_i = b_1\wedge b_2\wedge\cdots\wedge b_k$ is being called a *k-blade*. A *unit n-blade* is often referred to as a *pseudoscalar* $e_1\wedge e_2\wedge\cdots\wedge e_n = \mathbf{i}_n$, where $e_i$ are *unit vectors*, i.e. $e_i\cdot e_j = \delta_{ij}$ .

**Geometric Product:** let $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathcal{G}_n$ be Geometric Algebra vectors, then the geometric product $\mathbf{ab}$ is simply the sum of the inner and outer product $\mathbf{ab} = \mathbf{a}\cdot\mathbf{b}+\mathbf{a}\wedge\mathbf{b}$ with the properties:

$$\begin{aligned}(\mathbf{ab})\mathbf{c} &= \mathbf{a}(\mathbf{bc})\\ \mathbf{a}(\mathbf{b}+\mathbf{c}) &= \mathbf{ab}+\mathbf{ac}\\ (\mathbf{b}+\mathbf{c})\mathbf{a} &= \mathbf{ba}+\mathbf{ca}\ .\end{aligned}$$

The sum of scalars, vectors and blades is denoted as *multivector*, especially every single scalar, vector and blade is a multivector too, i.e. the result of any afore mentioned product is a *multivector* in general.

**Rotations in $\mathcal{G}_n$:** let $e_1, e_2, \ldots, e_n$ be unit vectors in $\mathcal{G}_n$, then with $i \neq j$

$$(e_i\wedge e_j)(e_i\wedge e_j) = -1\ , \tag{1}$$

a proof is given in Appendix A.1.

So, similar to *Euler's Formula*, by substituting $e_1\wedge e_2 = i$, where $i$ is the imaginary unit, and computing *Taylor series expansion* of $exp(\phi(e_1\wedge e_2))$, or in shorthand notation $e^{\phi(e_1\wedge e_2)}$, we get

$$e^{\phi(e_1\wedge e_2)} = \cos(\phi) + (e_1\wedge e_2)\sin(\phi)\ , \tag{2}$$

which is the well known rotation operator (rotor) for complex numbers.

Unlike the case of complex numbers, for a general rotation in $\mathcal{G}_n$ of an arbitrary plane $L = \sum_{i=1}^{n}\sum_{j=i+1}^{n} k_{ij} e_i\wedge e_j$ with $0 \leq k_{ij} \leq 1$ and $\sum_{i,j} k_{ij} = 1$ a two-sided rotor is needed

$$\mathbf{a}_{rotated} = e^{\frac{-\phi}{2}L}\,\mathbf{a}\,e^{\frac{\phi}{2}L}\ . \tag{3}$$

This is given by the fact that a two-sided rotor has no effect on vectors perpendicular to the rotation plane, e.g.

$$e^{\frac{-\phi}{2}e_1\wedge e_2}\,e_3\,e^{\frac{\phi}{2}e_1\wedge e_2} = e_3\ . \tag{4}$$

a proof is given in Appendix A.2. A one-sided rotor does not have this property.

## 3.2 Fourier-Mellin Transform

In this Section we briefly introduce the application of the Fourier-Mellin Transform for registration of grayscaled images. All theorems and proofs according to the Fourier-Transform can be found in [10].

As in [10], we use the following definition of the Fourier-Transform of a function $f : \mathbb{R}^2 \to \mathbb{C}^2$

$$F(\chi,\xi) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-2\pi i(\chi x+\xi y)}\mathrm{d}x\mathrm{d}y\ ,$$

and the Inverse Fourier-Transform as

$$\hat{f}(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(\chi,\xi)e^{-2\pi i(\chi x+\xi y)}\mathrm{d}\chi\mathrm{d}\xi\ .$$

**Translation Invariance:** let $f_1$ and $f_2$ be two images with the following relation:

$$f_1(x,y) = f_2(x-t_x, y-t_y)\ , \tag{5}$$

i.e. moving $f_2$ by $t_x$ (right) and $t_y$ (down) will result in $f_1$. Further let $F_1$ and $F_2$ be the Fourier-Transforms of $f_1$ respective $f_2$, then by the shift-theorem both are related to each other by:

$$F_1(\chi,\gamma) = F_2(\chi,\gamma)\,e^{-2i\pi(\chi t_x+\gamma t_y)}\ , \tag{6}$$

where $i$ is the imaginary unit.

The latter equation can be rearranged to:

$$\frac{F_2^*(\chi,\gamma)\,F_1(\chi,\gamma)}{|F_2(\chi,\gamma)|^2} = e^{-2i\pi(\chi t_x+\gamma t_y)}\ , \tag{7}$$

where $F_2^*(\chi,\gamma)$ denotes the conjugate complex Value of $F_2(\chi,\gamma)$.

The right-hand side of (7) is a Fourier-Transform of the Dirac-Delta function[1], i.e.:

$$e^{-2\pi i(\chi t_x+\gamma t_y)} = \iint_{\mathbb{R}^2} \delta(x-t_x, y-t_y)\,e^{-2\pi i(\chi x+\gamma y)}\,\mathrm{d}x\mathrm{d}y\ .$$

Since the inverse Fourier-Transform gives us the Dirac Delta function $\delta(x-t_x, y-t_y)$, the position of the Dirac impulse gives us the values for $t_x$ and $t_y$, as can be seen in Figure 1.

[1] or to be more precise *Dirac Delta distribution*


(a) (b) (c)

**Figure 1:** According to Paragraph *Translation Invariance* in Section 3.2, (a) is $f_1$ and (b) is $f_2$ with the relation $f_1(x,y) = f_2(x-50, y-10)$, which means that shifting (b) 50 pixels right and 10 pixels down will result in (a) (the orign is the upper left corner and the image sizes are $128 \times 128$). The *Inverse Fourier Transform* of the left-hand side of equation (7) results in (c). The *Dirac Impulse p*, i.e. the white point, has got the coordinates $p = (50,10)^T$ .

**Rotation and Scale Invariance:** let $f_1$ and $f_2$ be two images with the following relation:

$$f_1(x,y) = f_2(x\cos(\alpha) + y\sin(\alpha), y\cos(\alpha) - x\sin(\alpha)) \ , \quad (8)$$

i.e. rotating $f_2$ by the angle $\alpha$ will result in $f_1$. Transforming from Cartesian $(x,y)$ to polar coordinates, the latter equation can be written as:

$$f_1 = f_2(r, \phi - \alpha) \ ,$$

and is syntactically similar to equation (5) now. So the value for $\alpha$ can be computed in the same way as explained previously.

Having not only a difference in rotation but also in homogeneous scaling, the relation is then:

$$f_1(x,y) = f_2(\frac{x}{s}\cos(\alpha) + \frac{y}{s}\sin(\alpha), \frac{y}{s}\cos(\alpha) - \frac{x}{s}\sin(\alpha)) \ ,$$

where $s$ is the so called scaling factor.

Transforming both sides of the above equation to polar coordinates results in

$$f_1(r,\phi) = f_2(\frac{r}{s}, \phi - \alpha) \ .$$

Computing the logarithm of $\frac{r}{s}$ gives the so called *log-polar coordinates* which leads to

$$f_1(\log(r), \phi) = f_2(\log(r) - \log(s), \phi - \alpha) \ , \quad (9)$$

the same structure as in (5), so that $\alpha$ and $\log(s)$, respective the scaling factor $s$, can be computed as described above.

The Fourier-Transform $F^l$ of $f(x,y)$ in log-polar coordinates can be rearranged as follows:

$$\begin{aligned} F^l(u_1,u_2) &= \int_0^{2\pi}\int_0^{+\infty} f(\log(r),\phi) e^{-2\pi i(u_1\log(r)+u_2\phi)} \mathrm{d}\log(r)\mathrm{d}\phi \\ &= \int_0^{2\pi}\int_0^{+\infty} f(\log(r),\phi) r^{-2\pi i u_1} e^{-2\pi u_2\phi} \mathrm{d}\log(r)\mathrm{d}\phi \\ &= \int_0^{2\pi}\int_0^{+\infty} f(\log(r),\phi) r^{-2\pi i u_1 - 1} \mathrm{d}r e^{-2\pi u_2\phi} \mathrm{d}\phi \ . \end{aligned}$$

Since the inner integral is a Mellin-Transform while the outer integral is a Fourier-Transform, the whole formula is often referred to as the *Fourier Mellin Transform*.

**Translation, Rotation and Scale Invariance:** let $f_1$ and $f_2$ be two images, with the relation

$$f_1(x,y) = f_2(sx\cos(\alpha) + sy\sin(\alpha) - t_x, sy\cos(\alpha) - sx\sin(\alpha) - t_y) \ ,$$

i.e. rotating $f_2$ by the angle $\alpha$, scaling by $s > 0$ and finally shifting it by $t_x$ and $t_y$ will result in $f_1$ .

The Fourier-Transforms of both sides are related to each other by

$$F_1(\chi,\gamma) = F_2(\frac{\chi\cos(\alpha) + \gamma\sin(\alpha)}{a}, \frac{\gamma\cos(\alpha) - \chi\sin(\alpha)}{a}) \frac{e^{-2\pi i(\chi t_x + \gamma t_y)}}{a^2} \ .$$

The factor $e^{-2\pi i(\chi t_x + \gamma t_y)}$ is a rotor, i.e. it changes only the orientation but not the magnitudes of $F_2$. So computing the magnitude spectra $M_1, M_2$ of $F_1$ and $F_2$ leads to

$$M_1(\chi,\gamma) = M_2(\frac{\chi\cos(\alpha) + \gamma\sin(\alpha)}{s}, \frac{\gamma\cos(\alpha) - \chi\sin(\alpha)}{s}) \frac{1}{s^2} \ .$$

Fourier-Mellin Transforming both sides in log-polar coordinates, results in

$$F_{M_1}(\xi,\psi) = F_{M_2}(\xi,\psi) \frac{e^{-2\pi i(\xi s + \psi\alpha)}}{s^2} \ .$$

Since the divisor $s^2$ does not affect the position of the peak, it can be simply ignored, which leads finally to

$$F_{M_1}(\xi,\psi) \approx F_{M_2}(\xi,\psi) e^{-2\pi i(\xi s + \psi\alpha)} \ .$$

The latter equation is syntactically equal to eq. (6), i.e. the computation of $s$ and $\alpha$ can be done as describeb in the respective paragraph above.

After rotating and scaling the image with the computed values, the given images will differ in shift only, such that the computation of this value can be done as described previously.

# 4 GENERALIZED REGISTRATION

In this Section we generalize the method of registering grayscaled images as mentioned above, to a method that allows a registration of multichannel images as well. Therefore we make use of the Geometric Algebra and the *Clifford Fourier Transform* (CFT) introduced in [6] that is defined as

$$\mathcal{F}\{\mathbf{f}\}(\chi) = \int_{\mathbb{R}^n} \mathbf{f}(\mathbf{x}) e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} \mathrm{d}^n\mathbf{x} \ , \tag{10}$$

and its inverse

$$\mathcal{F}^{-1}\{\mathbf{f}\}(\mathbf{x}) = \int_{\mathbb{R}^n} \mathbf{f}(\chi) e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} \mathrm{d}^n\chi \ , \tag{11}$$

where $\mathbf{x}, \xi \in \mathbb{R}^n$ are two $n$ dimensional vectors, $\mathbf{i}_n$ is a *pseudoscalar*, i.e. $\mathbf{i}_n\mathbf{i}_n = -1$ (the proof is analogous to the one given in Appendix A.1), and $\mathbf{f}, \tilde{\mathbf{f}} : \mathbb{R}^n \to \mathcal{G}_m$ are multivector valued signals.

The main advantage of this Fourier Transform is that it is a sum of classical Fourier Transforms,

$$\begin{aligned}
\mathcal{F}\{\mathbf{f}\}(\chi) &= \int_{\mathbb{R}^n} \mathbf{f}(\mathbf{x}) e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} \mathrm{d}^n\mathbf{x} \\
&= \int_{\mathbb{R}^n} (f_1(\mathbf{x})\, e_1 + f_2(\mathbf{x})\, e_2 + \cdots + f_n(\mathbf{x})\, e_n)\, e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} \mathrm{d}^n\mathbf{x} \\
&= \int_{\mathbb{R}^n} (-1)^{n-1} f_1(\mathbf{x}) e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} e_1 \\
&\qquad + (-1)^{n-1} f_2(\mathbf{x}) e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} e_2 \\
&\qquad\quad + \cdots + (-1)^{n-1} f_n(\mathbf{x}) e^{-2\pi \mathbf{i}_n(\mathbf{x}\cdot\chi)} e_n \, \mathrm{d}^n\mathbf{x} \\
&= (-1)^{n-1} F_1(\mathbf{x}) e_1 + \cdots + (-1)^{n-1} F_n(\mathbf{x}) e_n \ ,
\end{aligned} \tag{12}$$

i.e. high-performance implementations can be realized by using Fast Fourier Transforms.

## 4.1 Translation Invariance

Let $\mathbf{f}_1, \mathbf{f}_2 : \mathbb{R}^n \to \mathbb{R}^m$ be two vector valued signals and let $\mathbf{x}, \mathbf{t} \in \mathbb{R}^n$ with the following relation:

$$\mathbf{f}_1(\mathbf{x}) = \mathbf{f}_2(\mathbf{x} - \mathbf{t}) \ .$$

The Fourier Transforms $\mathbf{F}_1, \mathbf{F}_2$ of $\mathbf{f}_1$ and $\mathbf{f}_2$ respectively are related by

$$\mathbf{F}_1(\chi) = \mathbf{F}_2(\chi) e^{-2\pi \mathbf{i}_n(\mathbf{t}\cdot\chi)} \ , \tag{13}$$

a proof is given in Appendix A.3.

Analogously to the afore mentioned case for grayscaled images, multiplying the inverse of $\mathbf{F}_2$ from the left and computing the inverse Clifford Fourier Transform (11) will result in the Dirac Delta function $\delta(\mathbf{x} - \mathbf{t})$, i.e. the impulse is located at position $\mathbf{t} \in \mathbb{R}^n$. A proof is given in A.4.

## 4.2 Rotation and Scale Invariance

As described in paragraph *Rotations in* $\mathcal{G}_n$ in Section 3.1, rotations are defined in unit planes $L = \sum_{i=1}^{n} \sum_{j=i+1}^{n} k_{ij} e_i \wedge e_j$ with $0 \le k_{ij} \le 1$ and $\sum_{i,j} k_{ij} = 1$. Obviously there are $\binom{n}{2}$ such planes that can be extracted as

$$e^{\phi L} = e^{\phi_1 e_1 \wedge e_2}\, e^{\phi_2 e_1 \wedge e_3} \ \ldots\ e^{\phi_{\binom{n}{2}} e_{n-1} \wedge e_n} \ ,$$

where $\phi = \frac{\phi_1}{k_{11}} + \frac{\phi_2}{k_{12}} + \cdots + \frac{\phi_{\binom{n}{2}}}{k_{n-1\,n}}$

This means that in $n$ dimensions two functions $\mathbf{f_1}, \mathbf{f_2} : \mathbb{R}^n \to \mathbb{R}^m$ that differ in rotation, can finally differ in at most $\binom{n}{2}$ angles. Taking this in mind one can see, that for computing all angles, the functions $\mathbf{f_1}, \mathbf{f_2}$ must have the form $\mathbf{f}(r, \phi_1, \phi_2, \ldots, \phi_{\binom{n}{2}})$, i.e. two equal functions that differ in rotation and scale (that is equal in all dimensions), $\mathbf{f}_1(\mathbf{x}) = \mathbf{f}_2(e^{-\frac{\alpha}{2}L}\, \frac{\mathbf{x}}{s}\, e^{\frac{\alpha}{2}L})$ can be written as

$$\mathbf{f}_1(\log(r), \phi) = \mathbf{f}_2(\log(r) - \log(s), \phi - \alpha) \ , \tag{14}$$

where $\phi = (\phi_1, \phi_2, \ldots, \phi_{\binom{n}{2}})^T$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{\binom{n}{2}})^T$ and $r > 0, s > 0$. These coordinates are related to the Cartesian coordinates by

$$\begin{pmatrix} r \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{\binom{n}{2}} \end{pmatrix} \widehat{=} \underbrace{e^{-\frac{\phi_1}{2} e_1 \wedge e_2} \ldots e^{-\frac{\phi_{\binom{n}{2}}}{2} e_{n-1} \wedge e_n} (r e_1) e^{\frac{\phi_1}{2} e_1 \wedge e_2} \ldots e^{\frac{\phi_{\binom{n}{2}}}{2} e_{n-1} \wedge e_n}}_{\text{Cartesian}} \tag{15}$$

Given equation (14), $s > 0$ and $\alpha \in \mathbb{R}^{\binom{n}{2}}$ can be computed in the same way as for shifted vector valued signals.

## 4.3 Translation, Rotation and Scale Invariance

Although so far the generalization was easily done by exchanging the classical Fourier-Transform by the so called Clifford-Fourier-Transform, the generalization of the registration of a rotated, scaled and shifted image is a challenging task.

Let $\mathbf{f}_1, \mathbf{f}_2 : \mathbb{R}^n \to \mathbb{R}^m$ be two multichannel images with the relation

$$\mathbf{f}_1(\mathbf{x}) = \mathbf{f}_2(e^{-\frac{1}{2}\phi L}\, \frac{\mathbf{x}}{s}\, e^{\frac{1}{2}\phi L} - \mathbf{t}) \ ,$$

with $\mathbf{t} \in \mathbb{R}^n, \phi \in \mathbb{R}^{\binom{n}{2}}$ and $s > 0$. The Fourier-Transforms of $\mathbf{f}_1$ and $\mathbf{f}_2$ have the relation

$$\mathbf{F}_1(\chi) = \mathbf{F}_2(e^{\frac{1}{2}\phi L}\, s\chi\, e^{-\frac{1}{2}\phi L})\, \frac{e^{-2\pi i_n \chi\cdot\mathbf{t}}}{s^n} \ .$$

Getting rid of the factor $e^{-2\pi i_n \chi\cdot\mathbf{t}}$ by computing the magnitudes, would allow to compute the scale and the angles and afterwards the shift. Unfortunately this step would result in a so called scalar field, which can be imagined as a grayscaled image, i.e. from here

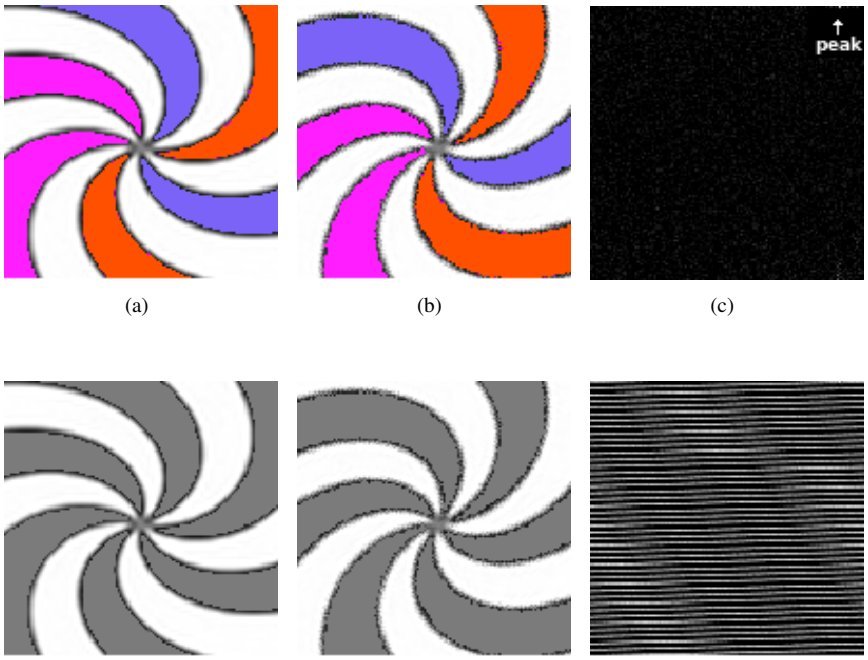



**Figure 2:** Images (a) and (b) are the two related color images that served as inputs for the presented multichannel registration algorithm. The result is shown in (c), a peak on the upper right. Transforming (a) and (b) into gray values (d) respective (e), the result (f) is no more a single peak, i.e. the rotation angle can not be computed anymore.

on the algorithm would operate on grayscaled and no more on multichannel images.

From equation (12) and the latter relation directly follows:

$$
\begin{aligned}
&(-1)^{n-1}F_{1_1}(\chi)e_1+(-1)^{n-1}F_{1_2}(\chi)e_2\cdots+(-1)^{n-1}F_{1_n}(\chi)e_n\\
&=F_{2_1}(e^{\frac{1}{2}\phi L}\,s\chi\,e^{-\frac{1}{2}\phi L})\,\frac{e^{-2\pi i_n\chi\cdot\mathbf{t}}}{s^n}e_1\\
&\quad+F_{2_2}(e^{\frac{1}{2}\phi L}\,s\chi\,e^{-\frac{1}{2}\phi L})\,\frac{e^{-2\pi i_n\chi\cdot\mathbf{t}}}{s^n}e_2\\
&\quad+\cdots+F_{2_n}(e^{\frac{1}{2}\phi L}\,s\chi\,e^{-\frac{1}{2}\phi L})\,\frac{e^{-2\pi i_n\chi\cdot\mathbf{t}}}{s^n}e_n\ ,
\end{aligned}
$$

where $F_{i_j}$ is the classical Fourier Transform of image $i$ and channel $j$. Computing the magnitudes component wise results in

$$
\begin{aligned}
&M_{1_1}(\chi)e_1+M_{1_2}(\chi)e_2+\cdots+M_{1_n}(\chi)e_n\\
&=M_{2_1}(e^{\frac{1}{2}\phi L}\,s\chi\,e^{-\frac{1}{2}\phi L})\,\frac{e_1}{s^n}+M_{2_2}(e^{\frac{1}{2}\phi L}\,s\chi\,e^{-\frac{1}{2}\phi L})\,\frac{e_2}{s^n}\\
&\quad+\cdots+M_{2_n}(e^{\frac{1}{2}\phi L}\,s\chi\,e^{-\frac{1}{2}\phi L})\,\frac{e_n}{s^n}\ ,
\end{aligned}
$$

or in shorthand notation

$$
\mathbf{M}_{F1}(\chi)=\mathbf{M}_{F2}(e^{\frac{1}{2}\phi L}\,\frac{\chi}{s}\,e^{-\frac{1}{2}\phi L})\frac{1}{s^n}\ ,
$$

which can be rewritten in transformed coordinates as

$$
\mathbf{M}_{F1}(\log(r),\phi)=\mathbf{M}_{F2}(\log(r)-\log(s),\phi-\alpha)\frac{1}{s^n}\ .
$$

Clifford Fourier Transforming both sides leads to

$$
\mathbf{F}_{M_{F1}}(\rho,\xi)=\mathbf{F}_{M_{F1}}(\rho,\xi)\frac{e^{-2\pi i_n\binom{\rho}{\xi}\cdot\binom{\log(s)}{\alpha}}}{s^n}\ .
$$

From here on, the computation of $s$, $\alpha$ and finally $\mathbf{t}$ can be done analogously to the case of grayscaled images, that was described previously.

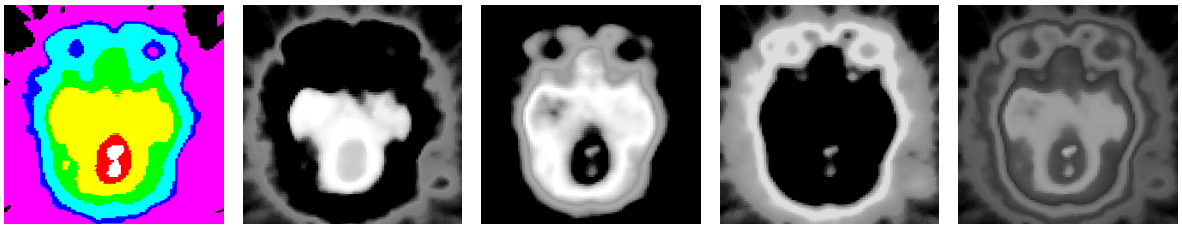
## 5 EVALUATION

The evaluation of our approach was made with two different kinds of data: artificial and real world data.

An artificial color image has been created such that the different colors within the image correspond to the same gray value after conversion to grayscale (cf. Figure 2). As real world data the 30th slice of a monkey head *Positron Emission Tomography* (PET) scan (cf. Figure 3), available from [13], has been chosen.

The registration of the artificial data was done by first rotating this image by a certain degree and then registering it as described in *Rotation and Scale Invariance* of Section 4. This step shows the potential and advantages of this multichannel registration and will be discussed in the next Section.

The evaluation of the monkey head PET was similar to the afore mentioned case of the artificial data. In contrast with the artificial data, the registration was executed on each single channel (red, green and blue), on the average signal of the three channels and on the



(a) (b) (c) (d) (e)

**Figure 3:** This images are taken from the 30th slice of a PET scan of a monkey head available from [13]. The images are respectively, the original PET image (a) itself, then the red (b), green (c) and blue (d) channels of the original image, and the average (e) of the three channels.

multichannel data itself. These steps were performed ten times to obtain many different results. Afterwards the min., max. and avg. errors were computed. As error measure we use the difference of the computed angle and the ground truth.

Since the registration on each single channel respectively on the average signal are *state of the art* approaches, this evaluation procedure compares them with the presented multichannel registration algorithm.

## 6 RESULTS

The basic outcome of this work is a novel approach for the registration of multichannel images, achieved by the generalization of the Fourier(-Mellin) Transform using Geometric Algebra multivectors as basis elements.

The main advantage of this algorithm is that it directly operates on the multichannel signal, instead of e.g. scaling the signal down to one dimension (e.g. by averaging) and thereby loosing a lot of information. This information loss in some cases will negatively affect on the accuracy of the registration results, as can be seen in Figure 2. Given the multichannel information, this artificially generated image is rotation invariant, i.e. the rotation relation of two different images (Figures 2(a) and (b)) is unique and can be computed with our algorithm. Computing the gray image would map all given colors, in this case, to the same gray values (Figures 2(d) and (e)), such that the computation of the angle is impossible from here on. Figures 2(c) and (f) depict the mentioned behavior of both processes, i.e. the former image shows a clearly visible peak on the upper right (the unique solution), while the latter image has got many different local maxima.

The PET images that were used for the evaluation are shown in Figure 3. Adding certain Gaussian noise ($\mu = 0$ and $\sigma^2 \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ ) to the rotated image and subsequent performing the steps as described in the previous Section, results in the computed min., max. and avg. errors that are shown in Tables 1,2 and 3 respectively.

| $\sigma^2$ | **ch. 1** | **ch. 2** | **ch. 3** | **avg.ch.** | **multich.** |
|---|---|---|---|---|---|
| 0 | 0.044 | 0.272 | 0.044 | 0.272 | 0.044 |
| 0.1 | 0.005 | 0.064 | 0.064 | 0.005 | 0.005 |
| 0.2 | 0.061 | 0.061 | 0.101 | 0.061 | 0.061 |
| 0.3 | 0.111 | 0.111 | 0.297 | 0.219 | 0.100 |
| 0.4 | 0.074 | 0.074 | 0.074 | 0.060 | 0.060 |
| 0.5 | 0.266 | 0.132 | 0.285 | 0.150 | 0.112 |

**Table 1:** min. errors

| $\sigma^2$ | **ch. 1** | **ch. 2** | **ch. 3** | **avg.ch.** | **multich.** |
|---|---|---|---|---|---|
| 0 | 2.189 | 2.030 | 3.436 | 2.189 | 1.062 |
| 0.1 | 1.624 | 1.447 | 2.994 | 1.955 | 1.248 |
| 0.2 | 99.738 | 2.128 | 4.405 | 4.118 | 1.612 |
| 0.3 | 88.483 | 4.405 | 4.108 | 161.608 | 1.806 |
| 0.4 | 48.528 | 3.430 | 31.653 | 155.403 | 1.762 |
| 0.5 | 79.519 | 55.548 | 118.452 | 134.363 | 1.891 |

**Table 2:** max. errors

| $\sigma^2$ | **ch. 1** | **ch. 2** | **ch. 3** | **avg.ch.** | **multich.** |
|---|---|---|---|---|---|
| 0 | 0.675 | 1.021 | 1.309 | 0.986 | 0.516 |
| 0.1 | 0.618 | 0.919 | 1.411 | 0.801 | 0.665 |
| 0.2 | 19.418 | 1.041 | 1.480 | 1.043 | 0.688 |
| 0.3 | 9.600 | 2.170 | 1.392 | 29.621 | 0.891 |
| 0.4 | 7.943 | 1.127 | 4.230 | 45.112 | 0.791 |
| 0.5 | 16.028 | 8.471 | 36.845 | 23.743 | 0.900 |

**Table 3:** avg. errors

These results show, that the registration performed on multichannel data (cf. column *multich.*) is more accurate and more stable to noise than the registration on each channel separately (cf. columns *ch. i*) or on the average of all channels (cf. column *avg.ch.*). At the same time one can see that the registration on channel 2 (blue) was far more accurate than the registration on the other single channels in the most cases, i.e. combining the results of the different channels, to compute the rotation angle, is a hard and still unsolved task. Having the multichannel registration as presented here, the solution of the mentioned task becomes no longer necessary since the registration is performed on all channels simultaneously.

# 7 CONCLUSION

The proposed registration algorithm is an extension of the well-known and widely used Fourier Transform which operates on scalar data (e.g. gray images). As many image sources provide multi-channel information, there is a need to process this data in an adequate way which prevents loss of information and keeps the original dimensionality of the information.

Geometric Algebra has been chosen as a mathematical framework which allows a.o. an extension of the Fourier Transform, called the Clifford Fourier Transform [6], to be able to operate on multichannel signals directly, i.e. it allows a frequency analysis on vector valued signals and an implementation of vector valued filters.

It has been shown in [6], that the Clifford Fourier Transform is a sum of many classical Fourier Transforms. As nowadays signal-processing hardware or recent graphics hardware provides interfaces to perform the Fast Fourier Transform (FFT) hardware-accelerated, it is suitable to design more complex algorithms on base of the FFT and remain in suitable computing times.

First investigation of the results shows that our registration on multichannel signals is very robust to noise, and that this approach provides potential for many applications on vector valued data.

# 8 FUTURE WORK

Our next steps will be a qualitative comparison with conventional approaches as well as an evaluation on volumetric image data.

Since a very high accuracy is needed, especially in the context of medical imaging, our algorithm will be extended with a spectrum based subpixel registration approach like [17], to achieve even more accurate results.

Having this, other approaches, such as the improvement of the resolution by registration [9], will be easily applicable with our multichannel registration algorithm as well.

Further, to improve performance, our future work will also include an implementation in CUDA respective OpenCL.

# REFERENCES


[1] Eduardo Bayro-Corrochano and Garret Sobczyk, editors. *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser, 2001.

[2] Thomas Bülow and Gerald Sommer. Hypercomplex signals-a novel extension of the analytic signal to the multidimensional case. *IEEE trans. on Signal Processing*, pages 2844–2852, 2001.

[3] Qin-Sheng Chen. *Image Registration and its Applications in Medical Imaging*. PhD thesis, Free University Brussels (VUB), 1993.

[4] Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists*. Cambridge University Press, 2003.

[5] J. Ebling and G. Scheuermann. Clifford Convolution And Pattern Matching On Vector Fields. In *Proceedings of IEEE Visualization(VIS)*, pages 193–200, 2003.

[6] Julia Ebling. Clifford fourier transform on vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):469–479, 2005.

[7] Todd A. Ell and Stephen J. Sangwine. Hypercomplex fourier transforms of color images. In *in Proc. ICIP*, pages 137–140, 2001.

[8] Michael Felsberg. *Low-Level Image Processing with the Structure Multivector*. PhD thesis, Inst. f. Informatik u. Prakt. Math. der Christian-Albrechts-Universität zu Kiel, 2002.

[9] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *CVGIP: Graph. Models Image Process.*, 53(3):231–239, 1991.

[10] Robert J. Marks II. *Handbook of Fourier analysis and its applications*. Oxford: Oxford University Press., 2009.

[11] Christian Perwass. *Geometric Algebra with Applications in Engineering*. Springer, 2009.

[12] D. L. Pham, C. Xu, and J. L. Prince. A survey of current methods in medical image segmentation. In *Annual Review of Biomedical Engineering*, volume 2, pages 315–338. 2000.

[13] Stefan Roettger. The erlangen volume library. `http://www9.informatik.uni-erlangen.de/External/vollib/`.

[14] Gustavo K. Rohde, Sinisa Pajevic, Carlo Pierpaoli, and Peter J. Basser. A comprehensive approach for multi-channel image registration. In *WBIR'03*, pages 214–223, 2003.

[15] Didier Stricker. Tracking with reference images: a real-time and markerless tracking solution for outdoor augmented reality applications. In *VAST '01: Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, pages 77–82, New York, NY, USA, 2001. ACM.

[16] Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, 2006.

[17] K. Takita, T. Aoki, Y. Sasaki, T. Higuchi, and K. Kobayashi. High-accuracy subpixel image registration based on phase-only correlation. In *IEICE Trans. Fund.*, volume E86-A, no. 8, pages 1925–1934, 2003.

[18] John A. Vince. *Geometric Algebra for Computer Graphics*. Springer, 1 edition, 4 2008.

[19] B. Zitova. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, October 2003.

# A MATHEMATICAL PROOFS

## A.1 Proof of (1)

$$\begin{aligned}(e_i \wedge e_j)(e_i \wedge e_j) &= (-e_j \wedge e_i)(e_i \wedge e_j)\\ &= (-e_j e_i)(e_i e_j)\\ &= -e_j \underbrace{e_i\, e_i}_{1} e_j\\ &= -\underbrace{e_j e_j}_{1}\\ &= -1\end{aligned}$$

q.e.d.

## A.2 Proof of (4)

$$\begin{aligned}&e^{\frac{-\phi}{2}e_1\wedge e_2}\, e_3\, e^{\frac{\phi}{2}e_1\wedge e_2}\\ &= (\cos(\frac{\phi}{2}) - (e_1\wedge e_2)\sin(\frac{\phi}{2}))\, e_3\, (\cos(\frac{\phi}{2}) + (e_1\wedge e_2)\sin(\frac{\phi}{2}))\\ &= (\cos(\frac{\phi}{2}) - (e_1\wedge e_2)\sin(\frac{\phi}{2}))(\cos(\frac{\phi}{2}) + (e_1\wedge e_2)\sin(\frac{\phi}{2}))\, e_3\\ &= (\cos(\frac{\phi}{2})^2 + \sin(\frac{\phi}{2})^2) e_3\\ &= e_3\end{aligned}$$

q.e.d.

## A.3 Proof of (13)

Let $\mathbf{f} : \mathbb{R}^n \to \mathcal{G}_n$ be a (multi-)vector valued function, then the Clifford-Fourier-Transform of $\mathbf{f}(\mathbf{x}-\mathbf{t})$ equals

$$\begin{aligned}\mathbf{F}_t(\mathbf{u}) &= \int_{\mathbb{R}^n} \mathbf{f}(\mathbf{x}-\mathbf{t}) e^{-2\pi i \mathbf{u}\cdot\mathbf{x}} \mathrm{d}^n\mathbf{x}\\ &= \int_{\mathbb{R}^n} \mathbf{f}(\hat{\mathbf{x}}) e^{-2\pi \mathbf{i}_n \mathbf{u}\cdot(\hat{\mathbf{x}}+\mathbf{t})} \mathrm{d}^n\hat{\mathbf{x}}\\ &= \int_{\mathbb{R}^n} \mathbf{f}(\hat{\mathbf{x}}) e^{-2\pi \mathbf{i}_n \mathbf{u}\cdot\hat{\mathbf{x}}} \mathrm{d}^n\hat{\mathbf{x}}\, e^{-2\pi \mathbf{i}_n \mathbf{u}\cdot\mathbf{t}}\\ &= \mathbf{F}(\mathbf{u})\, e^{-2\pi \mathbf{i}_n \mathbf{u}\cdot\mathbf{t}}\end{aligned}$$

where $\mathbf{F}$ is the Fourier Transform of $\mathbf{f}$ (providing that the Fourier-Transform exists).

Now let $\mathbf{f}_1, \mathbf{f}_2 : \mathbb{R}^n \to \mathcal{G}_n$ be two (multi-)vector valued functions, such that $\mathbf{f}_1(\mathbf{x}) = \mathbf{f}_2(\mathbf{x}-\mathbf{t})$. Let $\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_t$ be the Fourier-Transforms of $\mathbf{f}_1(\mathbf{x}), \mathbf{f}_2(\mathbf{x})$ and $\mathbf{f}_2(\mathbf{x}-\mathbf{t})$, respectively, then

$$\begin{aligned}\mathbf{F}_1(\mathbf{u}) &= \mathbf{F}_t(\mathbf{u})\\ &= \mathbf{F}_2(\mathbf{u})\, e^{-2\pi \mathbf{i}_n \mathbf{u}\cdot\mathbf{t}}\end{aligned}$$

q.e.d.

## A.4 Proof of Dirac Delta CFT

Let $\delta : \mathbb{R}^n \to \{0, +\infty\}$ be a Dirac Delta function, defined as

$$\delta(\mathbf{x}) = \begin{cases} +\infty & ,\mathbf{x} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \\ 0 & ,\text{else} \end{cases}$$

s.t. the constraint

$$\int_{\mathbb{R}^n} \delta(\mathbf{x})\, \mathrm{d}^n\mathbf{x} = 1 \tag{16}$$

The Clifford Fourier Transform of $\delta(\mathbf{x}-\mathbf{t})$ is then

$$\begin{aligned}\mathbf{F}(\mathbf{u}) &= \int_{\mathbb{R}^n} \delta(\mathbf{x}-\mathbf{t}) e^{-2\pi \mathbf{i}_n \mathbf{x}\cdot\mathbf{u}} \mathrm{d}^n\mathbf{x}\\ &= \int_{\mathbb{R}^n} \delta(\hat{\mathbf{x}}) e^{-2\pi \mathbf{i}_n (\hat{\mathbf{x}}+\mathbf{t})\cdot\mathbf{u}} \mathrm{d}^n\hat{\mathbf{x}}\\ &= \int_{\mathbb{R}^n} \delta(\hat{\mathbf{x}}) e^{-2\pi \mathbf{i}_n \hat{\mathbf{x}}\cdot\mathbf{u}} \mathrm{d}^n\hat{\mathbf{x}}\, e^{-2\pi \mathbf{i}_n \mathbf{t}\cdot\mathbf{u}}\\ &= e^{-2\pi \mathbf{i}_n \mathbf{t}\cdot\mathbf{u}}\end{aligned}$$

q.e.d.

# Fast Approximation of the Shape Diameter Function

Maurizio Kovacic
Dept. Mathematics & Computer Science
University of Cagliari
Via Ospedale, 72
09124, Cagliari (Italy)
mau.kovacic@gmail.com

Fabio Guggeri
Dept. Mathematics & Computer Science
University of Cagliari
Via Ospedale, 72
09124, Cagliari (Italy)
guggeri@unica.it

Stefano Marras
Dept. Mathematics & Computer Science
University of Cagliari
Via Ospedale, 72
09124, Cagliari (Italy)
stefano.marras@unica.it

Riccardo Scateni
Dept. Mathematics & Computer Science
University of Cagliari
Via Ospedale, 72
09124, Cagliari (Italy)
riccardo@unica.it

**ABSTRACT**

In this paper we propose an optimization of the Shape Diameter Function (SDF) that we call Accelerated SDF (ASDF). We discuss in detail the advantages and disadvantages of the original SDF definition, proposing theoretical and practical approaches for speedup and approximation. Using Poisson-based interpolation we compute the SDF value for a small subset of randomly distributed faces and propagate the values over the mesh. We show the results obtained with ASDF versus SDF in terms of timings and error.

**Keywords:** Segmentation, Poisson equation.

## 1 INTRODUCTION

The Shape Diameter Function (SDF) [18] has proven very useful for mesh skeletonization and segmentation. Closely related to the Medial Axis Transform (MAT) [5], it defines a scalar function over the points of a mesh representing the diameter of the shape's volume at each point while being computationally lightweight as compared to the MAT. The main contribution of the SDF is its taking into account the interior of the mesh and its volumetric information as opposed to the majority of segmentation algorithms that rely on local surface features as curvature; for each primitive it computes the diameter of the object along its interior, the result is a pose invariant function of the local volume that yields a good mesh partitioning. Moreover, it defines a set of internal points for each primitive, halfway through its interior, that can be used for skeleton extraction. In this paper we focus on optimization for segmentation purposes: a study of the behavior of the SDF function over the mesh suggests that, as main variations occur only in a small subset of the faces, it is possible to lower the number of computations with little to no effect on the final result by means of a constrained Poisson interpolation over the mesh (see figure 1 for an example). We will present a summary of the published works related to our problem in Section 2; Section 3 will contain a detailed description of the original SDF algorithm, while details on the Poisson Equation used to efficiently interpolate the values can be found in Section 4, with results and comparison between the optimized version and the original implementation in Section 5. Section 6 contains our conclusions and suggestions for future works.

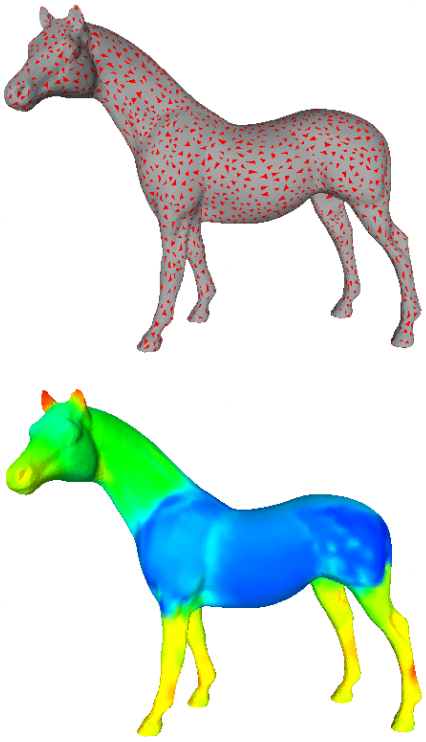


**Figure 1: The SDF function is computed only on the selected faces (red in the top image). The constrained interpolation results in a smooth and accurate descriptor (bottom image).**

## 2 RELATED WORK

### 2.1 Mesh Segmentation

There is a large amount of research works that study segmentation with many different strategies aiming to extract a semantically consistent partitioning. Most of the approaches rely on local surface properties for segmentation, as geodesic distance, angular distance or normal direction. In [12] the mesh is partitioned using the minima rule, that states that human perception divides shapes along the concave discontinuities; a snake contour is relaxated along the surface until convergence using geometric features as curvature and centricity. Minima rule is used also in [16] where the authors define an algorithm called Fast Marching Watersheds that identifies the regions bounded by contours of negative curvature; a similar technique can be found in [14] where an extension of the 2D morphological watershed segmentation over the mesh is used to partition the shape according to curvature. Katz and Tal, in [10], adopt the strategy to decompose the mesh in a hierarchical manner using fuzzy clustering according to a combination of surface metrics and using minimum graph cuts to extract the patch boundaries. Cohen-Steiner and colleagues in [8] propose a framework for shape approximation based on shape partitioning, obtaining a face clustering that minimizes an error metric based on normal deviation. These surface features may provide good results, however their sensitiveness on local surface variations makes them unsuitable for a pose invariant segmentation. The main advantage of the SDF algorithm is that it takes advantage of the connection between surface and volume of the object; such kind of approach can be found in [15], where the mesh is intersected with a sphere over each vertex and the behavior of the intersections is used for partitioning, or in [3] where geometric primitives are iteratively fitted to the mesh.

### 2.2 Applications of the Poisson Equation in Computer Graphics

The Poisson equation (PE) has been used in a large number of application areas. In computer graphics and related fields, one of the main applications of Poisson equation is the image processing. Perez and colleagues in [17] used Poisson equation in order to modify the content of an image; particularly, the gradient of the original image is computed, then modified according to desired result, and finally Poisson equation is solved to obtain a new image. Perez uses this approach in order to enhance the original image, by correcting local illumination or changing local color by defining appropriate constraint fields; the solution of the PE will be the image that best fit these constraint fields. In image processing, the Poisson equation has been used also for seamless image mosaic (as presented in [13], [20] and [19]), since the Poisson-based interpolation guarantees a smooth transition between two images (a property that makes Poisson equation also suitable for photomontage, in [1]). Poisson equation has been used also in geometry processing and mesh editing system. For example, in [22], gradient fields are used in order to model coordinate functions, and mesh editing is performed by locally adapting the gradients and solving the Poisson equation for the new coordinate functions. In this way, it is possible to perform operations such as deformation, merging, smoothing and denoising. Alexa [2] uses discrete Laplace and Poisson models to perform mesh editing and detail transfer from one mesh to another, while Xu et al. [21], in order to realize shape interpolation, use PE to find an intermediate shape that allows a smooth transition between the source shape and the target shape. PE has been also used in order to reconstruct the surface of a mesh starting from a point cloud as shown in [11] by Kazhdan et al.
Also, there is a number of definition and formulation of the Laplace-Beltrami operator, such as in [4], [6] or [7]; in this work, we propose an implementation of the Laplace-Beltrami operator which is slightly different from the previous ones but that showed to perform well and to be easy to implement.

## 3 THE SHAPE DIAMETER FUNCTION

The main idea behind the Shape Diameter Function is to take into account the volumetric information of the shape by defining a scalar function on the mesh representing the diameter of the interior of the object, similarly to the MAT where the scalar field represents the distance between each point to the nearest boundary point. However, while the MAT is computationally expensive and requires a discretization of the space, the SDF results in a faster and more robust descriptor.

Given a mesh M the SDF is a scalar function on the surface ($f_p : M \rightarrow \mathbb{R}$) defined as the neighborhood diameter of the object at each surface point $p \in M$. Such diameter is extracted by casting a cone of rays from the point $p$ to the interior of the mesh according to the inverse normal at $p$ and computing the distance between $p$ and each intersection between the rays and the mesh. In order to improve the robustness of this approach, *false* intersections are removed from the computation: those intersection points whose normal is in the *same direction* as

the point $p$, that is when the angle between the normals is less than 90° are not considered in the final computation. The definition is extremely simple and intuitive, however it is highly sensitive to noise and local variations; further improvement is obtained by considering just those rays whose length fall within a standard deviation from the median of all lengths in order to remove spurious intersections. The final value of the SDF is a weighted average of the remaining lengths: the weights are the inverse of the angle between the ray to the center of the cone, due to the fact that rays with larger angles are much more frequent and therefore must have smaller importance in the final averaging.

The authors show that the best results are obtained by casting 30 rays into a cone of 120° per point; smaller angles don't discriminate between the object parts and are extremely sensitive to local features, whilst larger angles cause some rays to intersect unrelated parts of the mesh and add errors to the computation. In our paper we stick to this parameters to produce comparable results with the ones provided in the original algorithm.

The algorithm as defined doesn't guarantee pose invariance. The authors propose a small number of bilateral filtering steps in order to reduce the variation of the SDF value after a pose change; one may refer to the original paper for the formulation of the filtering as it is unnecessarily verbose for our purposes. As for the intersection search, an Octree is used to spatially index the elements of the mesh for a reduced number of ray-triangle intersection tests.

# 4 ACCELERATED SDF (ASDF) VIA POISSON INTERPOLATION

The main contribution of our optimization method is based on an observation of the behavior of the SDF function for regular meshes. On simple meshes the difference between the SDF value of a primitive (face or vertex) and the SDF value of its neighborhood is approximately zero for primitives in the same *part* of the object and it increases smoothly on the boundaries of the *parts*. This means that little to none additional information is obtained by the SDF computation for a vertex or face whose neighbors have already been evaluated. Furthermore, the bi-lateral filtering step proposed in the original paper lowers the importance of a single computation in the final output. Figure 2 shows the normalized, absolute value Laplacian of the SDF computed for each face according to the formula:

**Figure 2: The absolute differences in SDF value between each face and its neighborhood, coded from blue where close to zero to red where maximum, shows that the function has very small variations on most of the surface**

$$F(p) = |\sum_{v \in N(p)} w_v SDF(p) - \sum_{v \in N(p)} w_v SDF(v)|$$

where $N(p)$ is the neighborhood of $p$ and $w_v$ is the weight of face $v$ defined as 1 over the distance between the barycenter of $v$ and the barycenter of $p$. In the image, the color red means zero or close to zero while blue is the maximum difference. It is worth to notice that the difference in SDF is higher where the shape changes sharply: thus, for segmentation purposes, it is possible to approximate the SDF on the whole mesh by propagating the function value computed on a small subset of faces. The mean of propagation we choose is solving a Poisson equation with Dirichlet boundary conditions, a technique that obtained good results in mesh editing [22] and image processing [17] under similar circumstances. This technique allows to easily compute a constrained interpolation over the mesh guaranteeing computational efficiency and robustness of the results.

## 4.1 Poisson Equation

The formulation of the Poisson equation that we use is defined as follows:

$$\Delta f = \nabla \mathbf{v} \quad \text{with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

where $f$ is an unknown scalar function, $\mathbf{v}$ a guidance vector field, $\nabla \mathbf{v}$ is the *divergence* of $\mathbf{v}$, $\Delta$ is

the Laplace operator and $f^*$ defines the values of a known scalar function at the boundary $\partial\Omega$ of a selected region $\Omega$. Solving this equation allows to reconstruct the unknown function by interpolating the boundary values so that the gradient of $f$ is as close as possible to the vector field $\mathbf{v}$, resulting in a smooth and seamless propagation that satisfies some user prescribed conditions; the unknown values in the user-selected region $\Omega$ are set to the known function $f^*$ in the border of $\Omega$ so that no seam is visible between the known and unknown regions, and the values change smoothly according to $\mathbf{v}$.

In our framework, the SDF is both the known and unknown scalar function and $\Omega$ is defined as the set of faces whose SDF hasn't been computed yet. $f$ consists of the known, exactly computed values of SDF where $f^*$ is the interpolated value of the SDF function where no actual ray casting will be performed. The choice of the guidance vector field for the interpolation is nontrivial: we want the Laplacian of the propagated SDF to be the same as the divergence of $\mathbf{v}$, thus we need to understand the behavior of the SDF function according to its neighborhood.

A good approximation of the divergence can then be obtained by the opposite of the curvature on each face $curv(p)$ (computed as the mean of its vertices' curvature): it is plausible to expect the diameter of a shape to slightly increase where the Gaussian curvature is less than zero, that is, where the normals of the faces converge and the neighborhood is concave. This assumption, while not taking into account the shape of the other side of the mesh, is still good enough for small neighborhoods.

The final formula for each unknown face is:

$$\sum_{v \in N(p)} w_v f(p) - \sum_{v \in N(p) \cap \Omega} w_v f(v) = \sum_{v \in N(p) \cap \partial\Omega} w_v f^*(v) - curv(p)$$

where the Laplacian is computed on the dual graph of the mesh. The above formulation causes each unknown SDF to be a function of the known values over the boundary and the local curvature, whereas a face with no known neighbors will obtain a value completely dependent on the curvature variation.

## 4.2 Face Selection

We did not point out how we select the faces over which we compute the SDF. In early stage of our development we were using fancy schemes for identifying the correct subset of set that were thought to maximize the *dispersion* of the selected faces over the mesh.

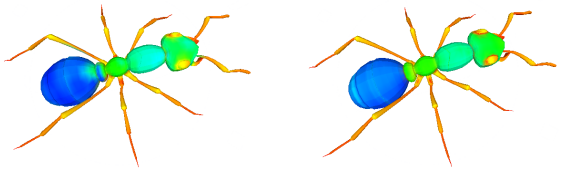


**Figure 3: SDF computation on the ant using 5% (left) and 100% (right) of the primitives, shown from red to blue according to the local thickness. It is noticeable how the presence of small local differences doesn't affect the global result with each segment easily distinguishable**

This choice had one major advantage: the deterministic choice of the face subset, but also a clear disadvantage since the selection stage was time consuming and the improvement over the original SDG was definitely moderate.

We then decided to follow a randomized scheme in selecting the faces. We, thus, compute an initial permutation of the face set and, in a second step, we select any single face in a simple manner, just picking every $n^{th}$ face and setting a flag on it identifying it as a *seed* for the solution of the Poisson equation. The mix of randomized input and Poisson equation revealed to be the best choice for our purposes.

# 5 RESULTS AND DISCUSSION

In this section we discuss the results obtained by our method in terms of time and error between the optimized and unoptimized version. For segmentation purposes it isn't mandatory that the SDF value of each single primitive is correct; in fact the segmentation process tends to split patches where the change rate is high on a significant area, while ignoring local peaks on the gradient. Therefore the correctness of a single face or vertex is discarded in favor of an overall correctness that is achieved by our propagation algorithm for a dense enough sampling. We can see in figure 3 that there is no substantial difference between the downsampled (5%) and original (100%); the differences in the values are restricted to a local point of view, whereas the global segmentation remains consistent.

To further discuss the relevance of these differences we show in figure 4 a map of the errors between a 10% subsampled and a complete SDF with blue being zero and red being the max difference. We can see how the highest differences are located along the boundaries of the mesh parts, due to the

sensitiveness of the original SDF definition on diameter variation.

However table 1 shows that the magnitude of this differences is low and doesn't strongly influence the outcome of the segmentation; moreover, the peaks in the errors occur on a small set of boundary faces, rapidly decreasing in their neighborhood: this may result in a fuzziness of the final segmentation border, with a negligible number of faces that are assigned to a patch that is different from the expected one, but still no substantial errors in the final segmentation.

As for the computational advantages of this optimization, we show in figure 5 a plot of the maximum and average error over the percentage of samples. Figure 6 shows the timings for the same computations. Timings don't reflect the ones presented in [18] and are obtained by an unoptimized single-thread implementation using the VCG library for the ray-triangle. We can anyway obtain an implementation independent speedup with a small error cost.

What is relevant of the ADSF can be understood looking with one eye at table 1 and the other at figure 5 reporting data about the same mesh: this will tell us that introducing less then 3% of error in the computation selecting only one face every tenth (as already mentioned this does not influence the overall segmentation at all) we gain one order of magnitude in the time spent for the computation passing from sixty to slightly more than six seconds.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper we presented our proposal for speeding-up an algorithm for mesh segmentation that uses the SDF function. We showed how the Poisson equation defined on the mesh vertices can be used to propagate, with a limited error, the value of a subsampled SDF by computing the function for a randomly distributed set of faces.

| Faces % | Max SDF error | Avg. SDF error |
|---|---|---|
| 50% | 0.4022 | 0.0099 |
| 20% | 0.5054 | 0.0198 |
| 10% | 0.5292 | 0.0272 |
| 5% | 0.5538 | 0.0374 |
| 2% | 0.5374 | 0.0575 |
| 1% | 0.5813 | 0.0695 |

**Table 1: Max and average error in SDF according to the percentage of faces used for computation. The number are normalized over the SDF values, so they ranges from 0 to 1. The computation was performed on the horse mesh.**

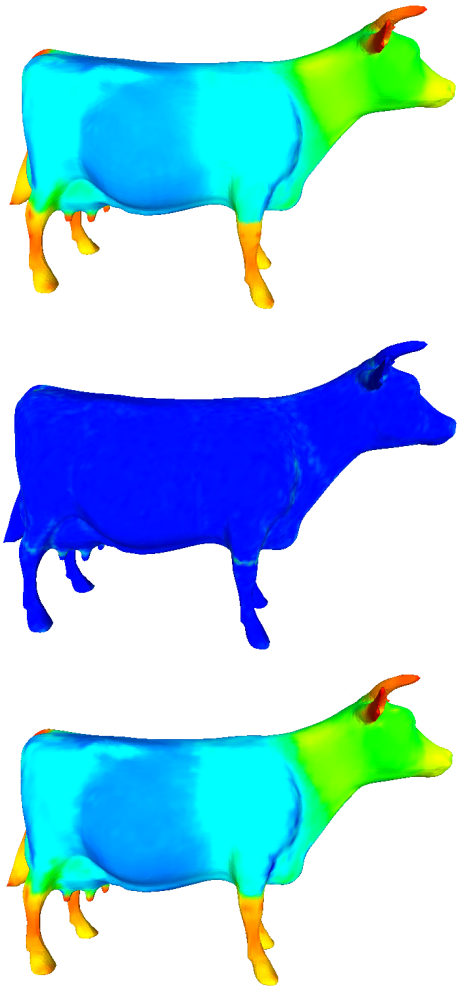


**Figure 4: In this picture you can visually appreciate the difference between the SDF computed on all the faces and a subset (10% of the total number). The image on top represents the result of the computation of the SDF on all the faces, while the one in the bottom represents the results of the computation performed on only 10% of the faces. In the middle image, the differences between the two results are graphically mapped on the mesh, with blue indicating no difference and colors towards red indicating larger and larger differences.**

The percentage of the samples can go down to 5% without sensible differences in the outcome.

A future improvement that we would like to explore is the choice of the samples according to their morphological significance instead of a random choice (see figure 7 for an example); we would like to study the behavior of our strategy if using a Gaussian sphere subsampling [9] where the samples are uniformly distributed over a Gaussian

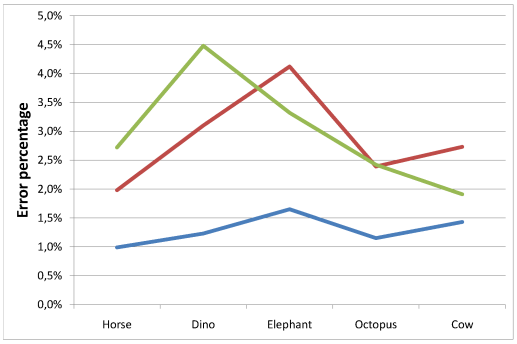



**Figure 5: Average error in SDF in function of the percentage of faces used for five different meshes. Green line are errors when selecting 10% of the faces, red line when selecting 20% and blue line when selecting 50%.**

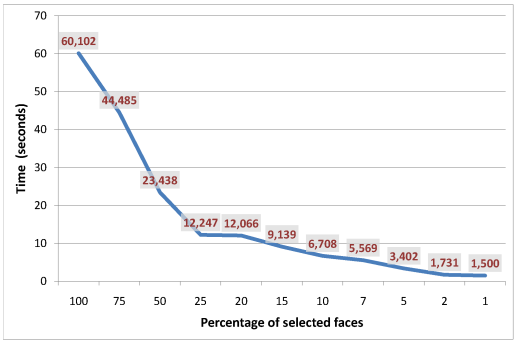



**Figure 6: Computational times in function of the number of selected faces. The times were taken while processing the horse mesh.**

sphere according to their normal resulting, therefore, more representative of the shape.

We also plan to improve the spatial organization for the intersection search: computing the ray-mesh intersection is a well known problem in Computer Graphics even outside the computational geometry area. In fact there is a lot of work on ray-tracing due to its centrality in rendering, and many of the techniques adopted in this field can be applied to the SDF problem in order to optimize the intersection search. While the original paper uses octrees as a mean of spatial indexing, it's reasonable to think that a KD-Tree can outperform it even when the data grow large or huge. It would be interesting to see how a specialized structure can further lower the computational times. One more open issue is how the parallelism implicit in ray-tracers can be exploited to work out a GPU implementation of the whole accelerated SDF. Furthermore, works on ray-tracing showed the usefulness of the SIMD paradigm with ray packing.

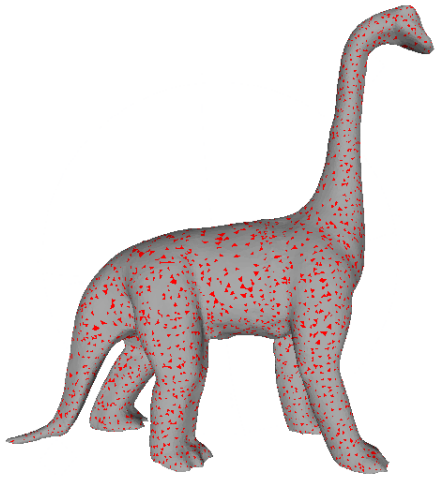


**Figure 7: Random face sampling (7%) - The quasi-uniform sampling gives no weight to the features of the mesh.**

## ACKNOWLEDGEMENTS

We would like to thank Lior Shapira for his suggestions and helpfulness.

## REFERENCES

[1] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 294–302, New York, NY, USA, 2004. ACM.

[2] Marc Alexa. Mesh editing based on discrete Laplace and Poisson models. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 51–59, New York, NY, USA, 2006. ACM.

[3] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22:181–193, 2006.

[4] Mikhail Belkin, Jian Sun, and Yusu Wang. Discrete Laplace operator on meshed surfaces. In *SCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 278–287, New York, NY, USA, 2008. ACM.

[5] H. Blum. A transformation for extracting new descriptions of shape. In *Models for the Perception of Speech and Visual Form*, pages 362–380, 1967.

[6] Alexander I. Bobenko. Delaunay triangulations of polyhedral surfaces, a discrete

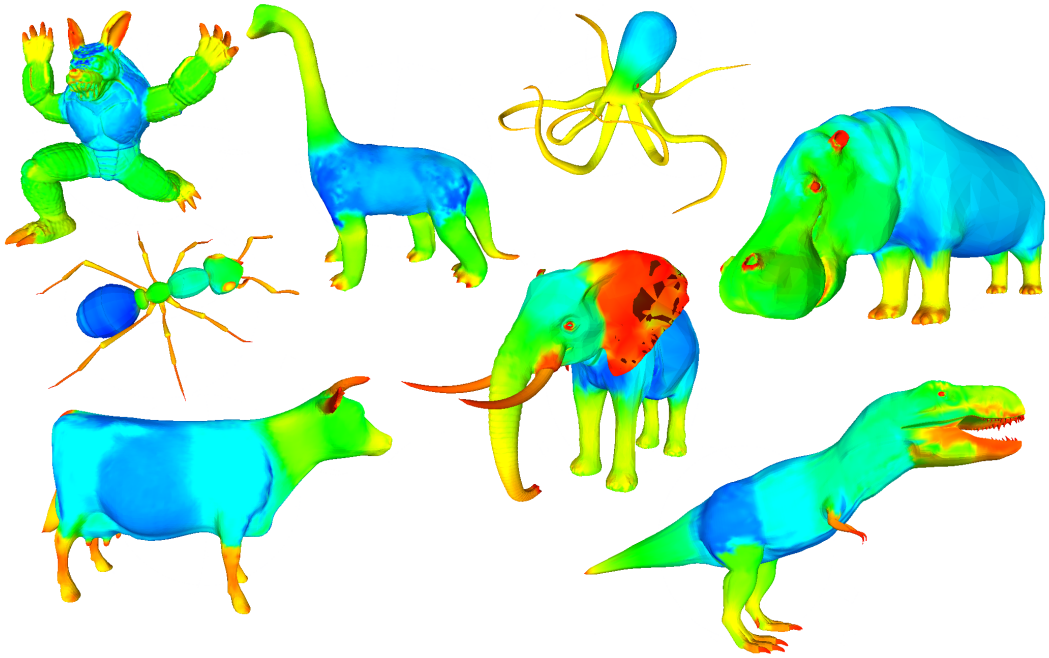


**Figure 8: Examples of application of ASDF to several different meshes. We preferred, for sake of understanding, here and in the rest of the paper, to show just the result of ASDF instead of the results of the segmentation phase.**

Laplace-Beltrami operator and applications. In *SCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 38–38, New York, NY, USA, 2008. ACM.

[7] Ming Chuang, Linjie Luo, Benedict J. Brown, Szymon Rusinkiewicz, and Michael Kazhdan. Estimating the Laplace-Beltrami operator by restricting 3D functions. *Computer Graphics Forum*, 28(5):1475–1484, July 2009.

[8] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 905–914, New York, NY, USA, 2004. ACM.

[9] Pablo Diaz-Gutierrez, Jonas Bösch, Renato Pajarola, and M. Gopi. Streaming surface sampling using gaussian $\varepsilon$-nets. *The Visual Computer*, 25:411–421, 2009.

[10] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 954–961, New York, NY, USA, 2003. ACM.

[11] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[12] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Intelligent mesh scissoring using 3D snakes. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 279–287, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless image stitching in the gradient domain. In *In Eighth European Conference on Computer Vision (ECCV 2004)*, pages 377–389. Springer-Verlag, 2003.

[14] Alan P. Mangan and Ross T. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[15] Michela Mortara, Giuseppe Patanè, Michela Spagnuolo, Bianca Falcidieno, and Jarek Rossignac. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. *Algorithmica*, 38(1):227–248, 2003.

[16] D. L. Page, A. F. Koschan, and M. A. Abidi. Perception-based 3d triangle mesh segmentation using fast marching watersheds. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, pages 27–32, June 2003.

[17] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 313–318, New York, NY, USA, 2003. ACM.

[18] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24:249–259, 2008.

[19] Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, 2006.

[20] Matthew Uyttendaele, Ashley Eden, and Richard Szeliski. Eliminating ghosting and exposure artifacts in image mosaics. In *2001 Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, pages 509–516, December 2001.

[21] Dong Xu, Hongxin Zhang, Qing Wang, and Hujun Bao. Poisson shape interpolation. *Graph. Models*, 68(3):268–281, 2006.

[22] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 644–651, New York, NY, USA, 2004. ACM.

# Operators for Multi-Resolution Morse Complexes in Arbitrary Dimensions

Lidija Čomić
Faculty of Engineering
University of Novi Sad
Trg D. Obradovića 6
Novi Sad, Serbia
comic@uns.ac.rs

Leila De Floriani
Dept of Computer Science
University of Genova
Via Dodecaneso 35
Genova, Italy
deflo@disi.unige.it

Federico Iuricich
Dept of Computer Science
University of Genova
Via Dodecaneso 35
Genova, Italy
federico.iuricich@gmail.com

**Abstract**

Ascending and descending Morse complexes, defined by the critical points and integral lines of a scalar field $f$ defined on a manifold $M$, induce a subdivision of $M$ into regions of uniform gradient flow, and thus provide a compact description of the morphology of $f$ on $M$. We propose a dual representation for the ascending and descending Morse complexes of $f$ in arbitrary dimensions in terms of an incidence graph. We describe atomic simplification and refinement operators on the Morse complexes and we investigate the effect of those operators on the graph-based representation of the two complexes. Simplification and refinement operators form a basis for a hierarchical multi-resolution representation of Morse complexes, from which it will be possible to dynamically extract representations of the morphology of the scalar field $f$ over $M$, at both uniform and variable resolutions.

## 1 Introduction

resenting morphological information extracted from discrete scalar fields is a relevant issue in several application domains, including terrain modeling, volume data analysis and visualization, and time-varying 3D scalar fields. Morse theory offers a natural and intuitive way of analyzing the structure of a scalar field $f$ as well as of compactly representing the scalar field through a decomposition of the domain of $f$ into meaningful regions associated with the critical points of the field. The ascending and the descending Morse complexes are defined by considering the integral lines emanating from, or converging to the critical points of $f$, while the Morse-Smale complex describes the subdivision of $M$ into parts characterized by a uniform flow of the gradient between two critical points of $f$.

Structural problems in Morse and Morse-Smale complexes, like over-segmentation in the presence of noise, or efficiency issues arising because of the very large size of the input data sets, can be faced and solved by defining simplification operators on such complexes and on their morphological representations.



Here, we present atomic operators for simplifying and refining Morse complexes. Such operators are defined in arbitrary dimensions and affect a constant number of entities in the Morse complexes. We show in [6] that the simplification operators together with their refinement ones define a basis for simplifying Morse (and Morse-Smale) complexes. Moreover, the general cancellation operator defined in Morse theory [19] can be expressed as a suitable combination of our operators.

We represent the ascending and descending Morse complexes as an incidence graph. This representation is based on encoding the incidence relations of the cells of the Morse complexes, and exploits the duality between the ascending and descending complexes. We define the effect of the simplification and of the refinement operators on the incidence-based dual representation of the descending and ascending Morse complexes. The two operators are defined in a dimension independent way, and their effect on the graph-based representation of the Morse complexes is easy to describe and implement. Moreover, they form the

basis for the definition of a hierarchical model of the Morse complexes. A hierarchical representation of the morphology of a scalar field is critical for interactive analysis and exploration in order to maintain and analyze characteristic features at different levels of abstraction.

The remainder of the paper is organized as follows. In Section 2, we review some basic notions on Morse theory and Morse complexes. In Section 3, we discuss some related work. In Section 4, we describe a dual incidence-based representation of the Morse complexes. In Sections 5 and 6, we present simplification and refinement operators respectively and we describe their effect on the incidence-based representation of the Morse complexes. Finally, in Section 7, we draw some concluding remarks and discuss current and future work.

# 2 Morse Theory and Morse Complexes

Morse theory studies the relationship between the topology of a manifold $M$ and the critical points of a scalar (real-valued) function defined on the manifold (for more details on Morse theory, see [19, 20]).

Let $f$ be a $C^2$ real-valued function defined over a closed compact $n$-manifold $M$. A point $p$ is a *critical point* of $f$ if and only if the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n})$ (in some local coordinate system around $p$) of $f$ vanishes at $p$. Function $f$ is a *Morse function* if all its critical points are non-degenerate (i.e. the Hessian matrix $Hess_p f$ of the second derivatives of $f$ at $p$ is non-singular). The number $i$ of negative eigenvalues of $Hess_p f$ is called the *index* of critical point $p$, and $p$ is called an *$i$-saddle*. A 0-saddle, or an $n$-saddle, is also called a *minimum*, or a *maximum*, respectively. An *integral line* of $f$ is a maximal path which is everywhere tangent to the gradient of $f$. Each integral line connects two critical points of $f$, called its *origin* and its *destination*.

Integral lines that converge to (originate at) a critical point $p$ of index $i$ form an $i$-cell ($(n-i)$-cell) $p$ called a *descending (ascending) cell*, or *manifold*, of $p$. The descending and ascending cells decompose $M$ into *descending* and *ascending Morse complexes*, denoted as $\Gamma_d$ and $\Gamma_a$, respectively, see Figure 1 (a) and (b) for a 2D example. We will denote as $p$ the descending $i$-cell of an $i$-saddle $p$. A Morse function $f$ is called a *Morse-Smale function* if the descending and the ascending manifolds intersect transversally. A *Morse-Smale complex* is defined by the connected components of the intersection

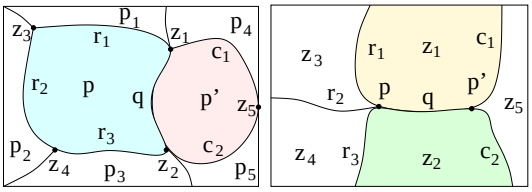



Figure 1: A portion of a descending Morse complex in 2D, with the descending cells of maxima $p$ and $p'$ highlighted (a), and the dual ascending Morse complex, with the ascending cells of minima $z_1$ and $z_2$ highlighted (b).

of descending and ascending Morse complexes. If $f$ is a Morse-Smale function, then complexes $\Gamma_a$ and $\Gamma_d$ are dual to each other.

# 3 Related Work

In this Section, we review related work on morphological representations of scalar fields provided by Morse or Morse-Smale complexes. We concentrate on two topics, which are relevant to the work presented here, namely: computation and simplification of Morse and Morse-Smale complexes.

Several algorithms have been proposed in the literature for decomposing the domain of a 2D scalar field $f$ into an approximation of a Morse, or a Morse-Smale, complex. Recently, some algorithms in higher dimensions have been proposed. For a review of the work in this area, see [3].

The extraction of critical points of a scalar field $f$ defined on a simplicial mesh has been investigated in 2D [2, 21], and in 3D [15, 25, 26, 24, 12] as a basis for computing Morse and Morse-Smale complexes. Algorithms for decomposing the domain $M$ of $f$ into an approximation of a Morse, or of a Morse-Smale complex in 2D can be classified as *boundary-based* [1, 4, 13, 22, 23], or *region-based* [5, 9]. In [12], an algorithm for extracting the Morse-Smale complex from a tetrahedral mesh is proposed. The algorithm, while interesting from a theoretical point of view, exhibits a large computation overhead, as discussed in [18].

Discrete methods rooted in the discrete Morse theory proposed by Forman [14] are computationally more efficient. In [9], a dimension-independent approach based on region growing has been proposed which implements the discrete gradient approach and computes the descending and the ascending Morse complexes. In [18], a region growing method, inspired by the watershed approach, has been proposed to compute the Morse-Smale complex. In [16], a Forman gradient

vector field $V$ is defined, and an approximation of the Morse-Smale complex is computed by tracing the integral lines defined by $V$.

One of the major issues that arise when computing a representation of a scalar field as a Morse, or as a Morse-Smale, complex is the over-segmentation due to the presence of noise in the data sets. *Simplification algorithms* have been developed in order to eliminate less significant features from a Morse-Smale complex. Simplification is achieved by applying an operator called *cancellation*, defined in Morse theory [19]. It cancels pairs of critical points of $f$, in the order usually determined by the notion of *persistence*, which is the absolute difference in function values between the paired critical points [13]. In 2D Morse-Smale complexes, the cancellation operator has been investigated in [4, 13, 17, 23, 27]. The cancellation operator on Morse-Smale and Morse complexes of a 3D scalar field has been investigated in [17] and [7], respectively.

# 4 A Dual Incidence-Based Representation for Morse Complexes

In this Section, we discuss a dual representation for the ascending and the descending Morse complexes $\Gamma_a$ and $\Gamma_d$, that we call the *incidence-based representation*. The underlying idea is that we can represent both the ascending and the descending complex as a graph by considering the boundary and co-boundary relations of the cells in the two complexes. In the discrete case, we consider a representation for the simplicial mesh $\Sigma$ which generalizes an indexed data structure commonly used for triangle and tetrahedral meshes [10], and we relate the two representations into the incidence-based data structure.

Recall that there is a one-to-one correspondence between $i$-saddles $p$ and $i$-cells $p$ in the descending complex $\Gamma_d$, and dual $(n-i)$-cells in the ascending complex $\Gamma_a$, $0 \leq i \leq n$. We exploit this duality to define a representation which encodes both the ascending and the descending complexes at the same time as an *incidence graph* [11]. The incidence graph encodes the cells of a complex as nodes, and a subset of the boundary and co-boundary relations between cells as arcs. The *incidence graph* associated with an $n$-dimensional descending Morse complex $\Gamma_d$ (and with an ascending Morse complex $\Gamma_a$) is a graph $G = (N, A)$, in which

1. the set of nodes $N$ is partitioned into $n+1$ subsets $N_0$, $N_1$,...,$N_n$, such that there is a one-to-one correspondence between nodes in $N_i$ (which we will call $i$-*nodes*) and the $i$-cells of $\Gamma_d$ (and thus the $(n-i)$-cells of $\Gamma_a$),
2. there is an arc joining an $i$-node $p$ with an $(i+1)$-node $q$ if and only if the corresponding cells $p$ and $q$ differ in dimension by one, and $p$ is on the boundary of $q$ in $\Gamma_d$ ($q$ is on the boundary of $p$ in $\Gamma_a$),
3. each arc connecting an $i$-node $p$ to an $(i+1)$-node $q$ is labeled by the number of times $i$-cell $p$ (corresponding to $i$-node $p$) in $\Gamma_d$ is incident to $(i+1)$-cell $q$ (corresponding to $(i+1)$-node $q$) in $\Gamma_d$.

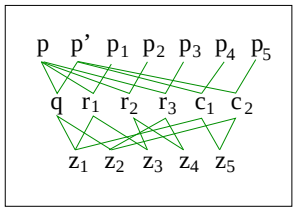



Figure 2: A portion of the incidence graph encoding the connectivity of descending and ascending Morse complexes illustrated in Figure 1 (a) and (b), respectively

Attributes are attached to the nodes of the incidence graph, containing information about geometry, and function values, while arcs have no associated (geometric) attributes. Note that the incidence graph provides also a combinatorial representation of the 1-skeleton of a Morse-Smale complex. Figure 2 shows a portion of the incidence graph encoding the connectivity of the descending Morse complex in Figure 1 (a), and of the ascending Morse complex in Figure 1 (b).

We have designed and implemented a data structure based on the incidence graph by encoding this latter as a standard adjacency list. We associate with each 0-node $p$ (corresponding to a minimum) a list of the $n$-simplexes of a simplicial complex $\Sigma$ forming the corresponding ascending $n$-cell of $p$. Dually, we associate with each $n$-node $p$ (corresponding to a maximum) a list of the $n$-simplexes forming the corresponding descending $n$-cell of $p$. The resulting data structure is the *incidence-based representation*.

# 5 Simplification Operators

In Morse theory, a general cancellation operator has been defined that allows eliminating any pair of critical points of consecutive index which are

connected by a unique integral line [19]. One of the drawbacks of such operator, when applied to a Morse-Smale complex, is that the number of cells in the complex can increase, and, when applied to a Morse complex, the number of incidences among cells can also increase.

In [8], we have defined two dual simplification operators in arbitrary dimensions, which we call *removal* and *contraction*. The two simplification operators are defined in a dimension-independent way. They are defined by imposing constraints on a cancellation operator, that allow us to avoid creating new cells in the Morse-Smale complex or new incidences in the Morse ones. The two operators form a complete set of basic operators for simplifying Morse complexes on a manifold $M$, as detailed in [6]. Moreover, the classical cancellation operator [19] can be seen as a macro-operator and expressed as a sequence of our atomic operators. A persistence value is associated with a simplification operator, and thus we apply simplifications in order of increasing persistence [13].

The first operator, called a *removal* of index $i$, $1 \leq i \leq n-1$, removes an $i$-saddle $q$ and an $(i+1)$-saddle $p$, provided that $q$ is connected by a unique integral line to an $(i+1)$-saddle $p$, and to exactly one other $(i+1)$-saddle $p'$ different from $p$, or to just one $(i+1)$-saddle $p$. In the first case, a removal of $q$ and $p$ is denoted as $rem(p, q, p')$, while in the second case as $rem(p, q, \emptyset)$. The second operator, that we call a *contraction* of index $i$, $1 \leq i \leq n-1$, removes an $i$-saddle $q$ and an $(i-1)$-saddle $p$ provided that $q$ is connected by a unique integral line to an $(i-1)$-saddle $p$, and to exactly one other $(i-1)$-saddle $p'$ different from $p$, or to just one $(i-1)$-saddle $p$. In the first case, a contraction of $q$ and $p$ is denoted as $con(p, q, p')$, and in the second case as $con(p, q, \emptyset)$. For the sake of simplicity, we discuss here only removals and contractions of the first kind.

## 5.1 Simplification on Morse complexes

The removal and contraction operators have a dual effect on the descending and the ascending Morse complexes. The effect of a contraction of index $i$ on $\Gamma_d$ ($\Gamma_a$) is the same as the effect of a removal of index $n-i$ on $\Gamma_a$ ($\Gamma_d$). For the sake of brevity, we describe the effect of the two operators on the descending Morse complex only.

The effect of a removal $rem(p, q, p')$ on the descending Morse complex $\Gamma_d$ is as follows: $i$-cell $q$, corresponding to $i$-saddle $q$ is deleted and $(i+1)$-cell $p$, corresponding to $(i+1)$-saddle $p$ is merged into $(i+1)$-cell $p'$, which corresponds to $(i+1)$-saddle $p'$. A contraction $con(p, q, p')$ deletes $i$-cell $q$ and merges $(i-1)$-cell $p$ into $(i-1)$-cell $p'$ in $\Gamma_d$. $i$-cell $q$ is contracted, and each $i$-cell in the co-boundary of $p$ is extended to include a copy of $i$-cell $q$, i.e., each $i$-cell in the co-boundary of $p$ is, after contraction, the union of itself with $i$-cell $q$.

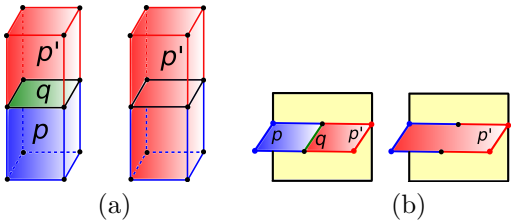


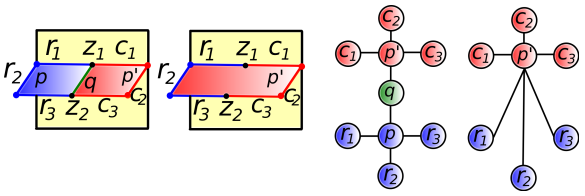

Figure 3: Portion of a 3D descending Morse complex before and after a removal $rem(p, q, p')$ of index 2 (a), and of index 1 (b).

The 2D case is simple, as our operators reduce to a minimum-saddle or a maximum-saddle cancellation operator. In 2D, there are exactly one removal and exactly one contraction operator (both of index 1). A removal deletes a 1-cell (saddle) $q$, and merges the two 2-cells (maxima) which shared $q$. It is the same as a maximum-saddle cancellation. A contraction contracts a 1-cell (saddle) $q$ and collapses the two 0-cells (minima) which bounded $q$. It corresponds to a minimum-saddle cancellation. Note that both operators involve an extremum and a saddle.

In 3D, there are two removal and two contraction operators. A removal of index 2 involves a 2-saddle $q$ and a maximum $p$ (it is a maximum-2-saddle cancellation). In the descending complex, it removes a 2-cell $q$, and merges 3-cell $p$ into a unique 3-cell $p'$ incident in $q$ and different from $p$, as illustrated in Figure 3 (a). A removal of index 1 involves a 1-saddle $q$ and a 2-saddle $p$. It is defined only if 1-cell $q$ is incident to exactly two different 2-cells $p$ and $p'$. It removes 1-cell $q$ and merges 2-cell $p$ into 2-cell $p'$, as illustrated in Figure 3 (b). Thus, it is a special case of a 1-saddle-2-saddle cancellation.

## 5.2 Simplification on the Incidence Graph

A removal, or contraction, simplification on the Morse complexes induces a modification on the incidence graph $G = (N, A)$ representing such complexes, that we call a *simplification modification*. Each simplification modification can be expressed as a deletion of two nodes $p$ and $q$ from $N$, and a replacement of a subset $A^+$ of the arcs in $A$ with another subset $A^-$ of arcs. For the sake of brevity, we will consider only a removal $rem(p, q, p')$ of in-




(a) (b)

Figure 4: Removal $rem(p, q, p')$ on a 3D descending Morse complex (a) and on the corresponding incidence graph (b).

dex $i$, $1 \leq i \leq n-1$.

Let $G = (N, A)$ be the incidence graph representing both the descending and the ascending Morse complexes $\Gamma_d$ and $\Gamma_a$ before a removal $rem(p, q, p')$. Then,

- $i$-node $q$ is connected through an arc in $A$ to exactly two different $(i+1)$-nodes $p$ and $p'$, such that the label of arcs $(q, p)$ and $(q, p')$ is 1, and to an arbitrary number of $(i-1)$-nodes from a set $Z = \{z_h, h = 1, .., h_{max}\}$;
- node $p$ is connected to an arbitrary number of $i$-nodes from a set $R = \{r_j, j = 1, .., j_{max} : r_j \neq q\}$, and to an arbitrary number of $(i+2)-$nodes from a set $S = \{s_k, k = 1, .., k_{max}\}$;
- node $p'$ is connected to an arbitrary number of $i$-nodes from a set $C = \{c_l, l = 1, .., l_{max} : c_l \neq q\}$, and to an arbitrary number of $(i+2)$-nodes from a set $D = \{d_m, m = 1, .., m_{max}\}$.

These conditions translate the feasibility condition of a removal operator.

For example, before the removal $rem(p, q, p')$, illustrated in Figure 4 (b), 1-node $q$ is connected to exactly two different 2-nodes $p$ and $p'$ (corresponding to 2-saddles), and to two 0-nodes $z_1$, and $z_2$ (corresponding to minima), which are not shown in the Figure. 2-node $p$ is connected to 1-nodes $r_1$, $r_2$ and $r_3$ and 2-node $p'$ is connected to 1-nodes $c_1$, $c_2$ and $c_3$. Nodes $p$ and $p'$ are connected to exactly the same 3-nodes $s_1$ and $s_2$, which are not shown in the Figure.

As an effect of a removal $rem(p, q, p')$ on $G$, nodes $p$ and $q$ are deleted, as well as all the arcs incident into $q$, and all the arcs incident into $p$ and connecting $p$ to $(i+2)$-nodes in $S$. All the arcs incident into $p$ and connecting $p$ to $i$-nodes in $R$ (with the exception of arc $(p, q)$), become incident in $p'$. Note that the effect of a contraction on $G$ is exactly the same as that of a removal, except for the fact that in a removal $q$ is an $i$-node, and $p$ and $p'$ are $(i+1)$-nodes, while in a contraction $q$ is an $i$-node and $p$ and $p'$ are $(i-1)$-nodes.

Thus, a simplification modification induced by a removal can be expressed as a local modification of the incidence graph $G = (N, A)$ which produces a graph $G' = (N', A')$, where:

- $N' = N \backslash \{p, q\}$, $A' = (A \backslash A^+) \cup A^-$, such that
  - $A^+ = \{(q, p)\} \cup \{(q, p')\} \cup \{(q, z_h) : z_h \in Z\} \cup \{(p, r_j) : r_j \in R\} \cup \{(p, s_k) : s_k \in S\}$,
    $A^- = \{(p', r_j), r_j \in R\}$.

Nodes $p$, $q$, $p'$, $z_h$, $r_j$ and $s_k$ are as described above. In addition, for each arc $(p, r_j)$, $r_j \neq q$, such that $(p', r_j)$ is also an arc in $A$ (i.e., such that $r_j = c_l$, for some $l$), the label of arc $(p', r_j)$ is increased by the label of arc $(p, r_j)$. A contraction can be expressed as a modification of graph $G$ in a completely dual fashion. Thus, a simplification modification on the incidence graph can be expressed as a pair $(A^+, A^-)$, i.e, as the collections of the arcs which are removed $(A^+)$ and which are inserted $(A^-)$.

In the example in Figure 4, after the removal of 1-saddle $q$ and 2-saddle $p$, nodes $q$ and $p$ are deleted from the incidence graph ($N' = N \backslash \{q, p\}$), arcs connecting $q$ to $p$ and $p'$, and arcs connecting 1-node $q$ to 0-nodes $z_1$ and $z_2$ (not illustrated in the Figure) are deleted, as are arcs connecting 2-node $p$ to 3-nodes $s_1$ and $s_2$ (not illustrated in the Figure). Arcs connecting 2-node $p$ to 1-nodes $r_1$, $r_2$ and $r_3$ are replaced by arcs connecting 2-node $p'$ to 1-nodes $r_1$, $r_2$ and $r_3$.

The effect on the incidence-based representation, that is the incidence graph extended with the references to the underlying simplicial decomposition, is restricted to the incidence graph when a simplification does not involve an extremum. When we perform a removal $rem(p, q, p')$ of index $n-1$, then the set of $n$-simplexes forming the descending cell of $p$ are merged into the set of $n$-simplexes forming the descending cell of $p'$. Dually, a contraction $con(p, q, p')$ of index 1 merges the $n$-simplexes of the ascending cell of $p$ with $n$-simplexes of the ascending cell of $p'$.

# 6 Refinement Operators

We have defined two refinement operators [6], which are inverse of the two simplification operators discussed in Section 5. Thus, they have the effect of introducing an $i$-saddle and an $(i+1)$-saddle by splitting an existing $i$-saddle or an $(i+1)$-saddle. They are defined as an undo of the corresponding simplifications. Before performing a refinement, the situation around the two newly introduced saddles, i.e., around the corresponding cells in the Morse complexes, needs to be the

same as it was at the time of the inverse simplification. Like the two simplification operators, the two refinement operators are dual to each other. The first operator, called an *insertion* of index $i$, splits an $(i+1)$-saddle $p'$ into $p'$ and an $(i+1)$-saddle $p$ by inserting an $i$-saddle $q$. The second operator, called an *expansion* of index $i$, splits an $(i-1)$-saddle $p'$ into $p'$ and an $(i-1)$-saddle $p$ by expanding an $i$-saddle $q$.

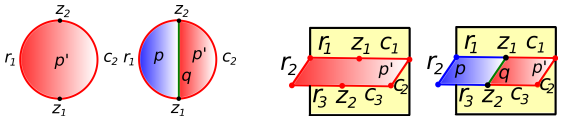
## 6.1 Refinement on Morse Complexes

In this Subsection, we discuss the effect of the refinement operators on the ascending and descending Morse complexes. In a descending complex $\Gamma_d$, an insertion of index $i$, denoted as $ins(p,q,p')$, which is the inverse (undo) of the removal $rem(p,q,p')$, consists of splitting an $(i+1)$-cell $p'$ into two new $(i+1)$-cells $p$ and $p'$, by inserting an $i$-cell $q$ into $(i+1)$-cell $p'$. $i$-cell $q$ is shared by $(i+1)$-cells $p$ and $p'$. For the correct application of the operator, we need to specify explicitly:

- the new cells $p$ and $q$, and the existing $(i+1)$-cell $p'$,
- $i$-cells $r_j$ in $R$, $j = 1,..,j_{max}$, which were on the boundary of $(i+1)$-cell $p'$ before the insertion, and which are on the boundary of $(i+1)$-cell $p$ after the insertion,
- $(i-1)$-cells $z_h$ in $Z$, $h = 1,..,h_{max}$, which are on the boundary of $i$-cell $q$ after the insertion, and
- $(i+2)$-cells $s_k$ in $S$, $k = 1,..,k_{max}$, which are in the co-boundary of $(i+1)$-cell $p$ after the insertion.

Note that the $(i+1)$-cells in the co-boundary of $i$-cell $q$ after the insertion are exactly $(i+1)$-cells $p$ and $p'$. The cells in $Z$ (which will be on the boundary of $i$-cell $q$), in $R$ (which will be on the boundary of $(i+1)$-cell $p$), and in $S$ (which will be in the co-boundary of $(i+1)$-cell $p$) need to be the same as the corresponding cells on the boundary and in the co-boundary of $p$ and $q$ before the inverse removal $rem(p,q,p')$.

Figure 5 (a) shows an insertion $ins(p,q,p')$ of index 1 of 1-cell $q$ and 2-cell $p$ into 2-cell $p'$ in a 2D descending Morse complex. It is specified by 1-cell $q$, 2-cells $p$ and $p'$, 0-cells $z_1$ and $z_2$ on the boundary of 1-cell $q$, and 1-cell $r_1$ on the boundary of 2-cell $p$. (The co-boundary of 2-cell $p$ in 2D is empty.) Figure 5 (b) shows the effect of the insertion $ins(p,q,p')$ of index 1 in a 3D descending Morse complex. It is specified by 1-cell $q$, 2-cells $p$ and $p'$, 0-cells $z_1$ and $z_2$ on the boundary of 1-cell $q$, 1-cells $r_1$, $r_2$ and $r_3$ on the boundary of 2-cell $p$, and 3-cells $s_1$ and $s_2$ in the co-boundary of 2-cell $p$.




(a) (b)

Figure 5: Insertion $ins(p,q,p')$ of index 1 on a descending Morse complex in 2D (a), and in 3D (b). It is specified by cells $p$, $q$ and $p'$, and cells in the immediate boundary and on the immediate co-boundary of the introduced cells $p$ and $q$.

An expansion of index $i$, denoted as $exp(p,q,p')$, which is the inverse of contraction $con(p,q,p')$, consists of splitting an $(i-1)$-cell $p'$ in $\Gamma_d$ into two new $(i-1)$-cells $p$ and $p'$ by expanding a new $i$-cell $q$ bounded by $p$ and $p'$. It is specified by a list of cells on the immediate boundary and on the immediate co-boundary of the new cells $p$ and $q$, which are the same as the corresponding cells before contraction $con(p,q,p')$.

## 6.2 Refinement on the Incidence Graph

Like a simplification operator on Morse complexes, a refinement operator on Morse complexes induces a modification on the incidence graph $G' = (N', A')$ representing these complexes, that we call a *refinement modification*. Since a refinement operator is defined as an undo of the corresponding simplification operator, each refinement modification on the incidence graph is also an undo of the corresponding simplification modification. A refinement modification can be expressed as an insertion of two nodes $p$ and $q$ into $N'$, and a replacement of a set $A^-$ of arcs in $A'$ with set $A^+$. Nodes $p$ and $q$ are the nodes which were eliminated by the inverse simplification modification, and $A^-$ and $A^+$ are exactly the same sets of arcs which defined the inverse simplification modification. In other words, a refinement modification inverse to a simplification modification defined by $(A^+, A^-)$ is defined by $(A^-, A^+)$.

Specifically, given an insertion operator $ins(p,q,p')$, the corresponding refinement modification of the incidence graph $G' = (N', A')$ produces a graph $G = (N, A)$, where:

- $N = N' \cup \{p, q\}$, $A = (A' \backslash A^-) \cup A^+$, where
  - $A^- = \{(p', r_j), r_j \in R\}$,
    $A^+ = \{(q,p)\} \cup \{(q,p')\} \cup \{(q,z_h) : z_h \in Z\} \cup$

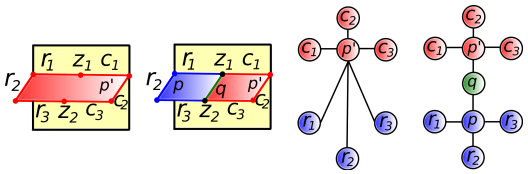



Figure 6: (a) Insertion operator $ins(p, q, p')$ of a 1-cell $q$ and 2-cell $p$ in the 3D descending complex, and (b) the corresponding refinement modification of the incidence graph.

$$\{(p, r_j) : r_j \in R\} \cup \{(p, s_k) : s_k \in S\}.$$

Here, $(i-1)$-nodes $z_h \in Z$ correspond to $(i-1)$-cells on the boundary of $i$-cell $q$, $i$-nodes $r_j \in R$ correspond to $i$-cells on the boundary of $(i+1)$-cell $p$, and $(i+2)$-nodes $s_k \in S$ correspond to $(i+2)$-cells in the co-boundary of $(i+1)$-cell $p$. Arc $(p', r_j)$ is removed from $A$ if label of arc $(p', r_j)$ minus label of arc $(p, r_j)$ equals 0. Otherwise, arc $(p', r_j)$ remains in $A$ with label diminished by label of arc $(p, r_j)$. An expansion can be expressed as a modification of graph $G'$ in a completely dual fashion. Figure 6 shows the effect of the refinement modification induced by an insertion $ins(p, q, p')$ of index 1 in 3D. Here, $Z = \{z_1, z_2\}$, $R = \{r_1, r_2, r_3\}$, and $S = \{s_1, s_2\}$.

# 7 Concluding Remarks

We have presented a dimension-independent representation which encodes both the ascending and descending Morse complexes in a single combinatorial structure, the incidence-based representation. This is achieved by exploiting the duality of the two complexes which leads to an incidence graph representation of their connectivity. We have described simplification operators for generalizing Morse complexes in arbitrary dimensions and their inverse refinement operators. In particular, we have presented their effect on the incidence graph in a completely dimension-independent way. The simplification and refinement operators are the basic ingredients for the definition of a hierarchical representation for the dual Morse complexes in terms of the incidence graph, which will provide a description of the Morse complexes at different levels of abstraction.

Currently, we are working on a dimension-independent implementation of simplification and refinement operators on the incidence-based representation. Our next step is the design and implementation of a multi-resolution representation for the two Morse complexes by defining its encoding data structure, an algorithm for computing it based on iterative simplification, and a selective refinement algorithm for extracting adaptive Morse complexes.

# Acknowledgements

This work has been partially supported by the National Science Foundation through grant CCF-0541032, and by MNTR of the Government of the Republic of Serbia through project 23036.

# References

[1] C. L. Bajaj and D. R. Shikore. Topology Preserving Data Simplification with Error Bounds. *Computers and Graphics*, 22(1):3–12, 1998.

[2] T. Banchoff. Critical Points and Curvature for Embedded Polyhedral Surfaces. *American Mathematical Monthly*, 77(5):475–485, 1970.

[3] S. Biasotti, L. D. Floriani, B. Falcidieno, P. Frosini, D. Giorgi, C. Landi, L. Papaleo, and M. Spagnuolo. Describing shapes by geometrical-topological properties of real functions. *ACM Comput. Surv.*, 40:Article 12, 2008.

[4] P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A Topological Hierarchy for Functions on Triangulated Surfaces. *Transactions on Visualization and Computer Graphics*, 10(4):385–396, July/August 2004.

[5] F. Cazals, F. Chazal, and T. Lewiner. Molecular Shape Analysis Based upon the Morse-Smale Complex and the Connolly Function. In *Proceedings of the nineteenth Annual Symposium on Computational Geometry*, pages 351–360, New York, USA, 2003. ACM Press.

[6] L. Čomić and L. De Floriani. Dimension-Independent Simplification and Refinement of Morse Complexes. *submitted*.

[7] L. Čomić and L. De Floriani. Cancellation of Critical Points in 2D and 3D Morse and Morse-Smale Complexes. In *Discrete Geometry for Computer Imagery (DGCI), Lecture Notes in Computer Science*, volume 4992, pages 117–128, Lyon, France, Apr 16-18 2008. Springer-Verlag GmbH.

[8] L. Čomić and L. De Floriani. Modeling and Simplifying Morse Complexes in Arbitrary Dimensions. In *Workshop on Topological Methods in Data Analysis and Visualization (TopoInVis'09)*, Snowbird, Utah, February 23 - 24 2009.

[9] E. Danovaro, L. De Floriani, and M. M. Mesmoudi. Topological Analysis and Characterization of Discrete Scalar Fields. In T.Asano, R.Klette, and C.Ronse, editors, *Geometry, Morphology, and Computational Imaging*, volume LNCS 2616, pages 386–402. Springer Verlag, 2003.

[10] L. De Floriani and A. Hui. Shape Representations Based on Cell and Simplicial Complexes. In *Eurographics 2007, State-of-the-art Report*. September 2007.

[11] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, Berlin, 1987.

[12] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale Complexes for Piecewise Linear 3-Manifolds. In *Proceedings 19th ACM Symposium on Computational Geometry*, pages 361–370, 2003.

[13] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse Complexes for Piecewise Linear 2-Manifolds. In *Proceedings 17th ACM Symposium on Computational Geometry*, pages 70–79, 2001.

[14] R. Forman. Morse Theory for Cell Complexes. *Advances in Mathematics*, 134:90–145, 1998.

[15] T. Gerstner and R. Pajarola. Topology Preserving and Controlled Topology Simplifying Multi-Resolution Isosurface Extraction. In *Proceedings IEEE Visualization 2000*, pages 259–266, 2000.

[16] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci. A Practical Approach to Morse-Smale Complex Computation: Scalability and Generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008.

[17] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. Topology-Based Simplification for Feature Extraction from 3D Scalar Fields. In *Proceedings IEEE Visualization'05*, pages 275–280. ACM Press, 2005.

[18] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann. Efficient Computation of Morse-Smale Complexes for Three-dimensional Scalar Functions. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1440–1447, 2007.

[19] Y. Matsumoto. *An Introduction to Morse Theory*, volume 208. American Mathematical Society, 2002.

[20] J. Milnor. *Morse Theory*. Princeton University Press, New Jersey, 1963.

[21] X. Ni, M. Garland, and J. C. Hart. Fair Morse Functions for Extracting the Topological Structure of a Surface Mesh. In *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH*, pages 613–622, 2004.

[22] V. Pascucci. Topology Diagrams of Scalar Fields in Scientific Visualization. In S. Rana, editor, *Topological Data Structures for Surfaces*, pages 121–129. John Wiley & Sons Ltd, 2004.

[23] S. Takahashi, T. Ikeda, T. L. Kunii, and M. Ueda. Algorithms for Extracting Correct Critical Points and Constructing Topological Graphs from Discrete Geographic Elevation Data. In *Computer Graphics Forum*, volume 14, pages 181–192, 1995.

[24] S. Takahashi, Y. Takeshima, and I. Fujishiro. Topological Volume Skeletonization and its Application to Transfer Function Design. *Graphical Models*, 66(1):24–49, 2004.

[25] G. H. Weber, G. Schueuermann, H. Hagen, and B. Hamann. Exploring Scalar Fields Using Critical Isovalues. In *Proceedings IEEE Visualization 2002*, pages 171–178. IEEE Computer Society, 2002.

[26] G. H. Weber, G. Schueuermann, and B. Hamann. Detecting Critical Regions in Scalar Fields. In G.-P. Bonneau, S. Hahmann, and C. D. Hansen, editors, *Proceedings Data Visualization Symposium*, pages 85–94. ACM Press, New York, 2003.

[27] G. W. Wolf. Topographic Surfaces and Surface Networks. In S. Rana, editor, *Topological Data Structures for Surfaces*, pages 15–29. John Wiley & Sons Ltd, 2004.

# Using Geometric Algebra for Visualizing Integral Curves

Werner Benger
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA-70803
werner@cct.lsu.edu

Marcel Ritter
Institute for Astro- and Particle Physics
University of Innsbruck
Innsbruck, A-6020
marcel@cct.lsu.edu

## ABSTRACT

The Differential Geometry of curves is described by means of the Frenet-Serret formulas, which cast first, second and third order derivatives into curvature and torsion. While in usual vector calculus these quantities are usually considered to be scalar values, formulating the Frenet-Serret equations in the framework of Geometric Algebra exhibits that they are best described by a bivector for the curvature and a trivector for the torsion. The bivector curvature field is directly suitable for visualization of integral curves for vector fields, providing "Frenet Ribbons" which are much richer in their visual expressiveness than lines. The set of quantities in the Frenet-Serret formalism allows to study numerical pitfalls for computing Frenet Ribbons. We show how to address them and demonstrate the applicability of the technique upon a complex numerical data set from computational fluid dynamics.

**Keywords:** Frenet Ribbon, pathline, streamline, computational fluid dynamics, curvature, torsion

## 1 INTRODUCTION

Numerical algorithms ultimately need to work with coordinates in the form of real numbers, thus $\mathbb{R}^n$. However, the early introduction of coordinates in the mathematical formulation of algorithms is, though common practice, highly problematic, as it obscures the view to the actual mathematical properties of the involved objects. Once an abstract mathematical object has been dismantled into numbers, even simple properties become complex. For instance, Sethian formulates the issue as "*the use of a coordinate system has nothing to do with the problem, but it has severely constrained our options*" [8] and Hermann Weyl wrote "*The introduction of numbers as coordinates by reference to the particular division scheme of the one-dimensional open continuum is an act of violence [...]*"[12]. In other words: while the algebraic operations on real numbers, the one-dimensional space $\mathbb{R}$, are part of common knowledge, it is a severe and unnecessary restriction to reduce other spaces – in particular $n$-dimensional manifolds such as used in geometry – to this set of one-dimensional algebraic operations (which is the process of introducing coordinates). Rather $n$-dimensional manifolds may carry an algebraic structure by themselves which should just be used, such as demonstrated by the coordinate-free approach of Geometric Algebra (GA) [2] which can directly be implemented using programming languages such as C++.

Curvature and torsion are common measures to express the properties of curves [7], as they represent the second and third derivative. In scientific visualization these measures serve to analyze the properties of streamlines in vector fields, but they can also be determined directly from the vector field itself, bypassing the process of computing an integral line [11].

Curvature and torsion are commonly introduced using vector algebra in the Euclidean space, involving formulations based on the cross-product. This formulation has several drawbacks: it hides the fact that torsion is a signed quantity, changing the sign under reflection; it relies on Cartesian coordinates, obscuring how to compute torsion with an explicit metric tensor such as required for curved space in relativity or curvilinear coordinates in computational fluid dynamics, and last not least formulations based on the cross-product do not generalize to higher dimension, which is required when we want to extend the formalism to study time-dependent vector fields in a four-dimensional framework. While there are $n$-dimensional vector calculus formulations of the Frenet-Serret formulae [7] available, GA improves the intuition of the involved objects.

The Frenet-Serret apparatus fails when the curvature becomes zero. Extensions to the Frenet frame have been proposed using quaternion formulations [6, 5]. In this paper we restrict ourselves to a review of the Frenet frame enhanced by a coordinate-free geometrical interpretation which is more intuitive than quaternions or vectors. The presented calculus is independent from the dimension of the underlying manifold and is expected to generalize to higher dimensions and treatment of parallel transport more intuitively in future work.



*GraVisMa 2010 conference proceedings,*
***GraVisMa'2010, Brno, 2010***
*Plzen, Czech Republic.*

# 2 MATHEMATICAL BACKGROUND

## 2.1 Differential Geometry

An $n$-dimensional manifold $M$ is a topological space that locally looks like $\mathbb{R}^n$. For each point there exists a neighborhood and a mapping, called a *chart* $\{x^\mu\} : M \to \mathbb{R}^n$. The transition from one chart $\{x^\mu\}$ to another chart $\{x^{\bar{\mu}}\}$ defines $n$ coordinate transformation functions $\{x^\mu\}\left(\{x^{\bar{\mu}}\}^{-1}\right) : X \to Y$ with $X, Y \subset \mathbb{R}^n$.If these are differentiable $k$ times, then the manifold is said to be $\mathscr{C}^k$. Space and time is modeled in physics as a $\mathscr{C}^\infty$-manifold. The laws of physics are independent of the choice of a coordinate system, which provide just representations of the mathematical objects.

**Curves vs. Lines** A *curve* is a mapping from a scalar $\lambda \in \mathbb{R}$, the curve parameter, to a point on a manifold: $q : \mathbb{R} \to M : \lambda \mapsto q(\lambda)$. The image of $q$ in $M$ is a *line*, a one-dimensional manifold. A certain line can be described by many curves which are distinct by different parameterizations. To describe a curve in a certain chart $\{x^\mu\}$ a coordinate function is used to extract a function for each coordinate of the chart:

$$\begin{array}{rcl} q^\mu : \mathbb{R} & \to & \mathbb{R} \\ \lambda & \to & q^\mu(\lambda) = x^\mu(q(\lambda)) \equiv x^\mu \circ q(\lambda) \end{array} \tag{1}$$

This set of $n$ functions $q^\mu(\lambda)$ is the representation of the curve $q$ in a coordinate system.

**Tangential Vectors** A tangential vector $v$ may be understood as a small displacement of neighboring points on a curve, where the components of the tangent vector are given by $v^\mu = dq^\mu(\lambda)/d\lambda$. Given a differentiable function $f : M \to \mathbb{R}$ we may evaluate it along the curve $f(q(\lambda)) : \mathbb{R} \to \mathbb{R}$ and find the directional derivative of $f$ along $q$ in a chart $\{x^\mu\}$ as

$$\frac{d}{d\lambda} f(q(\lambda)) = \frac{dq^\mu}{d\lambda} \frac{\partial f}{\partial x^\mu} \tag{2}$$

We may thus identify the tangential vector $v$ with the derivation operation $d/d\lambda$, which is an interpretation independent of any coordinate system:

$$v \equiv \frac{d}{d\lambda} = \frac{dq^\mu}{d\lambda} \frac{\partial}{\partial x^\mu} =: q'(\lambda) \tag{3}$$

The set of all derivatives $\frac{\partial}{\partial x^\mu}$ in all directions at a point $p$ defines the tangential space $T_p(M)$, which is a vector space (in contrast to $M$, which in general is not).

**Wedge (Outer) Product** The wedge product is denoted by the symbol "$\wedge$" and was introduced by Hermann Grassmann in the $19^{th}$ century. It allows to construct a vector space $\Lambda^2(T_p)$ from the tangential space $T_p$ by introducing the anti-symmetric (or wedge) product $\wedge : T_p \times T_p \to \Lambda^2(T_p)$ with $u, v \in T_p$, whereby $u \wedge v = -v \wedge u$. Higher orders of the form $\Lambda^k(T_p)$ with $k \leq dim(M)$ consist of so called *k-vectors* with *scalars* as *0-vectors*, *vectors* as *1-vectors*, *bivectors* as *2-vectors*, and so forth.

**Dot (Inner) Product** The *metric tensor field* is a scalar-valued symmetric bilinear function $g$ operating on tangential vectors, given at each point $p \in M$:

$$g : T_p(M) \times T_p(M) \to \mathbb{R} : u, v \mapsto g_p(u, v) \tag{4}$$

The metric tensor field allows to define the inner ("dot") product $u \cdot v := g_p(u, v)$ of two tangential vectors. The dot symbol "$\cdot$" is used by convention as a shortcut, but implying involvement of the metric tensor which needs to be explicitly specified for a manifold. In contrast, the wedge product is defined on the tangential space without any additional structure.

**Arc Length and Curve Tangent Vector** Arc length $s$ is defined as the length of integrated curve segments $s(\lambda) := \int_0^\lambda |q'(\tilde{\lambda})| \, d\tilde{\lambda}$ with $|v| = \sqrt{(g_p(v, v))}$. Derivation by the arc length will be denoted by dots:

$$\dot{v} := \frac{d}{ds} v \equiv \frac{d\lambda}{ds} \frac{d}{d\lambda} v = \dot{\lambda} \, v' \quad . \tag{5}$$

In general, derivation along a curve requires to employ a tangential transport and affine connection $\nabla$. It follows from (2.1) by derivation on both sides: $\frac{ds}{d\lambda} = |q'|$, which allows to express the derivation by arc length $s$ via the derivation by the curve parameter:

$$\dot{v} = \frac{1}{|q'|} \, v' \quad \text{or} \quad \frac{d}{ds} = \frac{d\lambda}{ds} \frac{d}{d\lambda} = \frac{1}{|q'|} \, \frac{d}{d\lambda} \tag{6}$$

Specifically it follows that the tangential vector $\dot{q}$ with respect to arc length - defined as the *tangent vector t* - is a unit vector $|\dot{q}| = 1$ due to $t := \dot{q} = \frac{1}{|q'|} \, q'$.

## 2.2 Geometric Algebra

Geometric Algebra is the generalization of vector calculus to form a complete set of algebraic operations on tangential vectors and $k$-vectors. Its central concept is the introduction of the invertible *geometric product*. Given two (tangential) vectors $u, v$ and a metric $g$, the requirements on the geometric product $uv$ are to be associative, left- and right-distributive and to fulfill $u^2 = uu = g(u, u)$. These postulates lead to the geometric product as $uv = u \cdot v + u \wedge v$ , which is now invertible.It is important to keep in mind that the geometric product is not commutative, thus in general $uv \neq vu$ and one needs to distinguish among left- and right-multiplication. As the geometric product sums a scalar value and a bivector it operates no longer on tangential vectors alone, but on the $2^n$-dimensional space of *multivectors* $V \in \bigoplus_{k=0}^n \Lambda^k(T_p)$

**Inner and Outer Product in GA** Expressing the inner and outer product via the geometric product may well lead to easier expressions since the geometric product is invertible and associative. For 1-vectors the inner

product is given by the symmetric part of the geometric product, the outer product as the anti-symmetric part:

$$u \cdot v \equiv \frac{1}{2}(uv + vu) \quad , \quad u \wedge v \equiv \frac{1}{2}(uv - vu) \quad . \tag{7}$$

Another useful operator, the Hodge-star operator $\star$, maps $k$-multivectors to $(n-k)$-vectors via the product with a pseudoscalar (an $n$-multivector) $\Omega \in \Lambda^n(T_p)$.

$$\star : \Lambda^k(T_p) \to \Lambda^{n-k}(T_p) : V \mapsto \Omega V \quad . \tag{8}$$

It allows to identify vectors and bivectors in three-dimensional space. For instance, the cross product in three-dimensional vector calculus corresponds to

$$u \times v \equiv \star(u \wedge v) \quad , \tag{9}$$

the difference being that "$\times$" is only defined in 3D, whereas the right side works in arbitrary dimensions.

**Vector Projections** Using the geometric product on two arbitrary vectors $u, v$ the expression $wuw$ with a unit vector $w = v/|v|$ yields the vector $u$ as *reflected* at the vector $v$. Adding the reflected vector $wuw$ to $u$ yields the component of the vector $u$ that is parallel to $w$:

$$u_{\parallel v} = \frac{1}{2}\left(u + \frac{vuv}{|v|^2}\right) \quad , \tag{10}$$

while subtraction yields the perpendicular component

$$u_{\perp v} = \frac{1}{2}\left(u - \frac{vuv}{|v|^2}\right) \quad , \tag{11}$$

where evidently $u = u_\parallel + u_\perp$. In GA (11) is called a *rejection* operation. Both components correspond to the inner and outer product (7) when multiplied with the inverse vector $v^{-1} \equiv v/|v|^2$:

$$u_{\parallel v} = \frac{(uv + vu)}{2}\,\frac{v}{|v|^2} = (u \cdot v)v^{-1} = v^{-1}(u \cdot v) \quad , \tag{12}$$

$$u_{\perp v} = \frac{(uv - vu)}{2}\,\frac{v}{|v|^2} = (u \wedge v)v^{-1} = -v^{-1}(u \wedge v) \quad . \tag{13}$$

**Relation to Vector Calculus** In 3D Euclidean space, we get the orthogonal component via the cross-product:

$$u_{\perp v} = \frac{v \times (u \times v)}{v^2} \tag{14}$$

Using the vector triple product formula relating cross and dot product $a \times (b \times c) = b(a \cdot c) - c(a \cdot b)$ we see

$$u_{\perp v} = \frac{u(v \cdot v) - v(u \cdot v)}{v^2} = u - (u \cdot v)v/v^2 \equiv u - u_{\parallel v}. \tag{15}$$

**Derivative of a Unit Vector** The derivative $d/d\lambda$, denoted by a prime as shortcut in the following expressions, of an (arbitrary) unit vector field $v/|v|$ along a curve yields a vector field that is orthogonal to the original vector field $v$:

$$\frac{d}{d\lambda}\frac{v}{|v|} = \frac{|v|\frac{d}{d\lambda}v - v\frac{d}{d\lambda}|v|}{|v|^2} \tag{16}$$

$$\frac{d}{d\lambda}v^2 = \frac{d}{d\lambda}vv = vv' + v'v = 2v \cdot v' \tag{17}$$

$$\frac{d}{d\lambda}|v| = \frac{d}{d\lambda}\sqrt{v^2} = \frac{v \cdot v'}{|v|} \tag{18}$$

therefore

$$\begin{aligned}\frac{d}{d\lambda}\frac{v}{|v|} &= \frac{1}{2|v|}\left(v' - \frac{vv'v}{|v|^2}\right) \overset{(11)}{=} v'_{\perp v}/|v| \\ &\equiv \frac{(v' \wedge v)\,v^{-1}}{|v|} \equiv \frac{(v' \wedge v)\,v}{|v|^3}\end{aligned} \tag{19}$$

i.e. the derivative of a unit vector field $v$ along a curve is perpendicular to the original field. A visualization of this behavior is shown later in Fig. 5. The same fact is evident from $v \cdot \frac{d}{d\lambda}\frac{v}{|v|} = 0$, noticing eqn. (18) becomes zero for a unit vector $|v| = 1$,

Consecutively applying the operations of derivation and normalization on the tangential vectors of a curve leads to a systematic scheme allowing to study a curve's properties, known as the Frenet-Serret formulas.

## 2.3 Frenet-Serret Formulae

**Curvature of a Curve** The curvature $\kappa$ of a curve is defined as the magnitude of the rate of change of the unit tangent vector $t$with respect to arc length:

$$\kappa := |\dot{t}| = \left|\frac{d}{ds}\,t\right| \equiv \frac{1}{|q'|}\,|t'| \tag{20}$$

The derivative of the tangent vector is perpendicular to $q'$ by means of (19):

$$t' = \frac{q''_{\perp q'}}{|q'|} \equiv \frac{(q'' \wedge q')\,q'}{|q'|^3} \quad . \tag{21}$$

The curvature can thus be seen as the rejection (perpendicular component) of the second derivative $q'' = d/ds\,q'$ by the velocity $q'$ normalized by the speed:

$$\kappa = \left|\frac{q''_{\perp q'}}{|q'|^2}\right| = \frac{|(q'' \wedge q')\,q'|}{|q'|^4} \tag{22}$$

By construction the curvature $\kappa$ is independent of the parameterization and is a measure that only depends on the line, as in (20) we differentiate with respect to arc length, not the curve parameter.

**Relation to Vector Calculus** By means of (14) we may express $t'$ in (21), and thus $\dot{t}$, as a cross product,

$$\dot{t} = \frac{q''_{\perp q'}}{|q'|^2} = \frac{(q'' \wedge q')\, q'}{|q'|^4} = \frac{q' \times (q'' \times q')}{|q'|^4} \qquad (23)$$

such that via $|a \times (b \times a)| = |a|\ |b \times a|$ we get the commonly shown formula for curvature as

$$\kappa = \frac{|q' \times (q'' \times q')\,|}{|q'|^4} = \frac{|q'|\ |q'' \times q'|}{|q'|^4} = \frac{|q'' \times q'|}{|q'|^3}\ . \qquad (24)$$

**Normal Vector and Osculating Bivector** Derivation and normalization of the tangential vector $t = q'/|q'|$ yields the *normal* unit vector, a quantity independent of the curve parameterization:

$$n := \frac{t'}{|t'|} \equiv \frac{\dot{t}}{|\dot{t}|} \overset{(20)}{\equiv} \frac{1}{\kappa}\dot{t} \overset{(6)}{=} \frac{1}{\kappa}\frac{1}{|q'|}t' \qquad (25)$$

By definition of the curvature (20) we trivially arrive at the first Frenet-Serret equation:

$$\boxed{t' = |t'|\ n = |q'|\ \kappa\, n} \text{ or } \boxed{\dot{t} = \kappa\, n} \qquad (26)$$

The tangent and normal vector define the *osculating plane* of the curve, called the binormal vector $t \times n$ in vector calculus. It corresponds to a bivector in Geometric Algebra ($t \cdot n = 0$):

$$b := tn = t \wedge n = -n \wedge t = -nt \quad . \qquad (27)$$

This "osculating bivector" $b$ is a unit bivector fulfilling $b^2 = -1$. The associated "curvature bivector" $\kappa b = t \wedge \dot{t}$ fulfills $(t \wedge \dot{t})^2 = -\kappa^2$.

**Relation to Vector Calculus** The normal vector expressed in derivatives of the curve $q$ becomes in Euclidean vector calculus, using (23) and (24):

$$n = \frac{q' \times (q'' \times q')}{|q'|\ |q'' \times q'|} \qquad (28)$$

**Torsion Trivector** The change of normal vector yields the form of a unit vector derivative (19):

$$n' = \frac{t''_{\perp t'}}{|t'|} \equiv \frac{(t'' \wedge t')\, t'^{-1}}{|t'|} \qquad (29)$$

To compute the change of the osculating bivector, we utilize the Leibniz rule on (27), notice that $t'$ and $n$ are parallel by eqn. (26) and reorder terms to find

$$\begin{aligned} \dot{b} &= \frac{1}{|q'|}\ \big(\underbrace{t' \wedge n}_{0} + t \wedge n'\big) = \frac{1}{|q'|}\ t \wedge \frac{(t'' \wedge t')\, t'^{-1}}{|t'|} \\ &= -\underbrace{\frac{1}{|q'|}\ \frac{t \wedge t' \wedge t''}{|t'|^2}}_{=:\tau}\ \underbrace{t'/|t'|}_{n} = -\frac{t \wedge \dot{t} \wedge \ddot{t}}{\kappa^2}\, n =: -\tau n \end{aligned} \qquad (30)$$

This is the third Frenet-Serret equation,

$$\boxed{b' = -\tau\, n\, |q'|} \text{ or } \boxed{\dot{b} = -\tau\, n} \qquad (31)$$

which in this formulation relates the change of the osculating bivector to the normal vector via the torsion trivector $\tau$. With the geometric product being invertible we can easily express $n$ by means of $b$ by noting $t^{-1} = t$ due to $|t| = 1$, finding $n = t^{-1}b = tb$. Derivation yields

$$\dot{n} = \dot{t}b + t\dot{b} = \kappa n\, b - t\ \tau n \qquad (32)$$

and using $nb = ntn = -nnt = -t$ with $t\tau = \tau\, t$ provides via $tn = b$ the second Frenet-Serret equation:

$$\boxed{\dot{n} = -\kappa\, t - \tau\, b} \qquad (33)$$

Here $b$ is a bivector (describing curvature) and $\tau$ is a trivector (describing torsion). It is evident that $\tau$ is not a scalar, but a pseudo-scalar - it changes sign under reflection: a helix with positive torsion seen in a mirror exhibits negative torsion.

**Relation to Vector Calculus** In a chart the torsion trivector will be expressed as a three-indexed object $\tau = \tau_{ijk}\ \partial_i \wedge \partial_j \wedge \partial_k$ . These are $2^n$ components, but in three dimensional Euclidean space they reduce to a single number and the torsion trivector can be associated with a scalar $|\tau|$ by means of the hodge-star operator as $\star\tau = |\tau|\Omega$. It expresses the torsion trivector relative to an orientation $\Omega$ describing the left-handedness or right-handedness of the chosen coordinate system. We can express eqn. (33) through the vector dual $\vec{b}$ to the bivector $b = \star\vec{b} = \Omega\vec{b}$, which due to $\Omega^2 = -1$ yields the usual Frenet-Serret equation for vectors:

$$\dot{n} = -\kappa\, t - |\tau|\Omega\ \ \Omega\vec{b} = -\kappa\, t + |\tau|\ \vec{b} \quad . \qquad (34)$$

# 3 VISUALIZING CURVES

## 3.1 Integral Curves

Given a vector field $v : M \to T(M) : q \mapsto v(q)$ a curve $q(\lambda)$ is an integral curve on this vector field if it fulfills $\frac{d}{d\lambda}q = v\big(q(\lambda)\big)$ . The properties of the integral curve are determined by the vector field itself. Curvature and torsion can be computed directly as curvature and torsion *fields* from the vector field [11] based on the vector field's Jacobian. As the previous has shown, these are actually bivector and trivector fields, not scalar fields.

## 3.2 Verification Vector Field

Verification of computational methods on behalf of analytical test data sets is of utmost importance. Here, a vector field is used for verifying the computational methods and is constructed to yield clearly defined results (stream lines) in the form of circles: $v = \partial_\varphi = (-y, x, 0)/\sqrt{x^2 + y^2}$. To check the independence of the

curve parameterization we provide a non-constant velocity depending on the angle relative to the coordinate system: $v = \left[1 + A \sin\left(\varphi \arctan \frac{y}{x}\right)\right] \partial_\varphi$ with $A$ an amplitude of the modification and $\varphi$ a modification factor for the angular dependency. With $A = 0.9$ and $\varphi = 1.0$ we get a vector field that is nearly zero for $y < 0$ and is large up to $|v| = 1.9$ for $y > 0$, as show in Fig. 1. The integral lines of this vector field are closed circles. To simulate data stemming from a numerical simulation the vector field is sampled to uniformly spaced locations and interpolated to arbitrary locations where the integral line passes through.

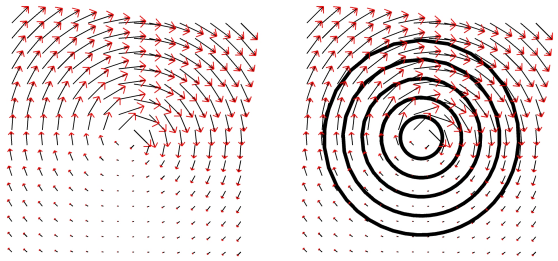


Figure 1: 2D slice of an axial vector field with non-constant (but non-vanishing) velocity sampled on a grid of $16^3$ points (left image). The integral lines of this vector field are closed circles (right image) of constant curvature, with curvature increasing toward the center.

## 3.3 Fields for Visual Analysis

The differential geometric treatment of curves systematically leads to a set of fields that allow to study the properties of a curve. Some of these fields are local quantities, i.e. they can be computed from the properties of a curve at each point and its neighborhood, but are otherwise independent of global quantities which depend on the entire shape of the curve. Both local and global quantities are of interest. Some of them are dependent on the parameterization and others are pure line quantities which do no change under re-parameterization.

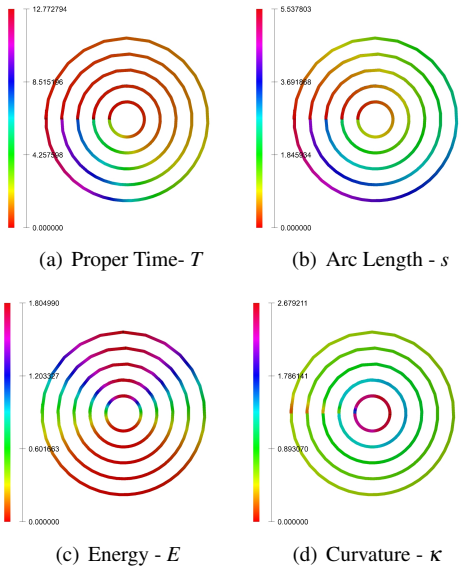
1. Proper time: $T = \int 1/|\dot{q}(\lambda)| d\lambda$
2. Arc length: $s = \int ds$
3. Velocity: $q'$
4. Coordinate Acceleration: $q''$
5. Energy: $E = |q'|^2/2$
6. Tangential vector: $t = q'/|q'|$
7. Normal vector: $n = t'/|t'|$
8. Osculating bivector: $b = tn$
9. Curvature: $\kappa = |\dot{t}| = |(q'' \wedge q')\, q'|/|q'|^4$
10. Curvature bivector $\kappa b = t \wedge \dot{t}$
11. Torsion trivector: $\tau = (t \wedge \dot{t} \wedge \ddot{t})/\kappa^2$
12. Torsion: $|\tau| = |\dot{b}| = |t \wedge \dot{t} \wedge \ddot{t}|/\kappa^2$

These quantities will be of type scalar, vector, bivector and trivector, each of these types requiring a different kind of visualization method along the curve. Scalar fields are commonly displayed via color-coding, vector fields via arrow icons. With bivectors and trivectors being elements of Geometric Algebra beyond the usual vector calculus, there are no common visualization methods for these types of fields. However, Frenet Ribbons, discussed in 3.4, provide a direct visualization of the osculating bivector field.

**Scalar Fields** The set of available scalar fields from the above set can be organized - for planar curves (zero torsion) - with respect to their properties of being local or global and their dependence on the curve parameterization (line quantity vs. curve quantity):

$$\begin{pmatrix} T & s \\ E & \kappa \end{pmatrix} \hat{=} \begin{pmatrix} global/curve & global/line \\ local/curve & local/line \end{pmatrix} \tag{35}$$

As demonstrated in Fig. 2, displaying these four quantities along a line provides four different views with complementary information. If we modify the input vector field by an arbitrary modulation of its amplitude, then the right column of Fig. 2 will remain unchanged, while only the left column will undergo changes. On the other hand, the lower row will be independent of the placement of integral seed points. Mapping proper



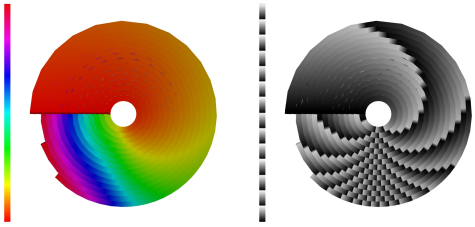

(a) Proper Time- $T$ (b) Arc Length - $s$

(c) Energy - $E$ (d) Curvature - $\kappa$

Figure 2: Visualization matrix of scalar fields on a curve: proper time $T$, arc length $s$, energy $E$ and curvature $\kappa$. Upper row are global (integral) quantities, right column are independent of parameterization.

time to colors provides a notion of the time that a particle requires to reach a certain point on this trajectory.

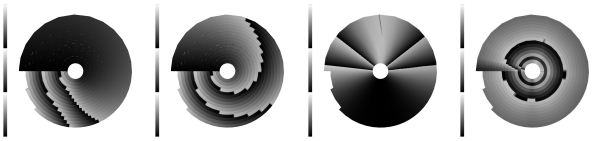
A colormap that uses gradient steps is furthermore able to emphasize the increments of proper time along the line, even in mere grayscale depiction. It provides a visual indication of the velocity and therefore the original vector field. Particles are traveling slower in the lower



(a) Value → color hue (b) Value → intensity steps

Figure 3: Visualizing a dense set of curves: Proper time with color map, showing advancement of time along the curves, and proper time with "zebra" colormap, depicting the velocity along the curve.

section of Fig. 3, which is conveyed better by the chosen gradient colorization. Fig. 4 shows the scalar fields with the "zebra" map applied.



(a) $T$ (b) $s$ (c) $E$ (d) $\kappa$

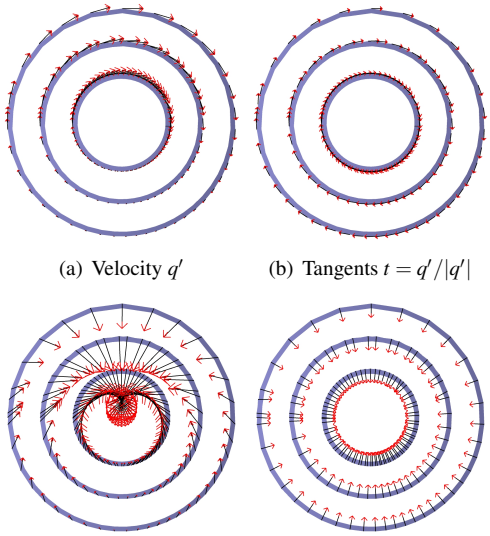
Figure 4: "Zebra" colorization scheme applied to the matrix of scalar fields (Fig. 2) for a dense set of curves: proper time, arc length, energy, curvature.

**Vector Fields** The velocity $q'$ along a curve is supposed to be identical with the value of a vector field $v(q(s))$ if $q$ is an integral curve. For vector fields given on discrete points its visualization may provide insight into the behavior of the interpolation algorithm, as discussed in 3.5, which in particular is non-trivial for curvilinear grids [9] [4] [10].

Same as with scalar fields, we can distinguish among curve and line quantities based on the dependency of a vector field on the curve parameterization. Since all vector fields "live" in the tangential space $T_p(M)$, they are local by nature. The notion of global vs. local is hereby replaced by order of derivation, firstly considering first and second order:

$$\begin{pmatrix} q' & t \\ q'' & n \end{pmatrix} \hat{=} \begin{pmatrix} 1^{st}/curve & 1^{st}/line \\ 2^{nd}/curve & 2^{nd}/line \end{pmatrix} \qquad (36)$$

The corresponding vector fields are shown in Fig. 5. Note that the acceleration $q''$, Fig. 5(c) is not normal to the velocity $q'$, Fig. 5(a), but lays in the same plane as the normal vectors $n$, Fig. 5(d). We can thus see Fig. 5 as a direct visualization of eqn. (21) which states that the direction of the normal vector is given by the projection of the acceleration on the velocity $n \propto q''_{\perp q'}$.



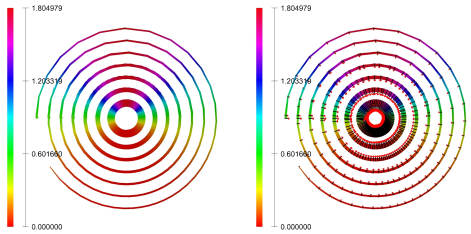
(a) Velocity $q'$ (b) Tangents $t = q'/|q'|$

(c) Acceleration $q''$ (d) Normals $n = t'/|t'|$

Figure 5: Visualization matrix of vector fields on a curve: velocity and acceleration (left column) depend on the curve's parameterization, tangents and normals (right column) are pure geometrical quantities. Upper row includes first order derivatives of the curve, lower row is based on second order derivations.

## 3.4 Visualizing the Curvature Bivector: Frenet Ribbons

A Frenet Ribbonis a direct visualization (Fig. 6) of the curvature bivector field $\kappa\ b = t \wedge \dot{t}$ along a curve $q$. The Frenet Ribbon is the surface generated by sweeping the tangential derivative vector $t'$ along the curve $q$. Its width depicts the curvature $\kappa = |\dot{t}|$, the location of the surface relative to the curve $q$ depicts the sign of the curvature, since osculating plane is described by the bivector $t \wedge \dot{t} = -\dot{t} \wedge t$.



(a) Frenet Ribbon (b) Ribbon and normals

Figure 6: A Frenet Ribbon is generated by sweeping the curve normal vectors along the curve. Colorization by energy.

Using modern graphics hardware, Frenet Ribbons are very suitable to be implemented using geometry shaders which allow generating the actual geometry completely on the GPU while just providing the line and normal vector information for each vertex. Consequently rendering Frenet Ribbons is about as fast as

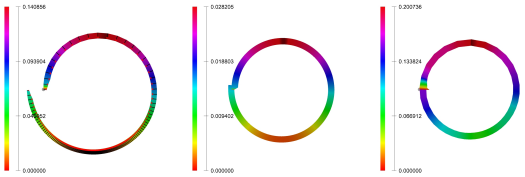


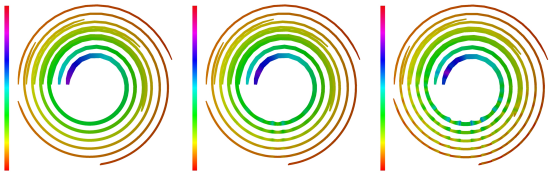
Figure 7: Accuracy and performance of integration methods: Euler integration with stepsize 1.0, 180 steps; stepsize 0.2, 720 steps; 8th order Runge-Kutta (DOP853), 40 steps.

drawing line primitives unless there occur huge polygons to be rendered due to locations of very high curvature. Usually rendering is possible in realtime with at least 30fps using a decent modern graphics card which supports geometry shaders.

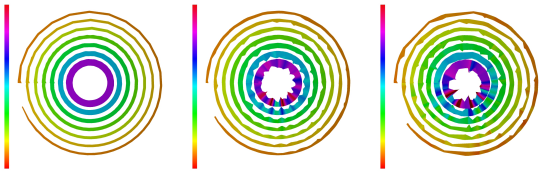
## 3.5 Numerics

**Integration Method** The forward Euler method is the most simple method to advance a point at a curve via $q(s+\Delta) = q(s) + \Delta\, v(q(s))$ for a constant step size $\Delta$. It is known to always gives overshoots of the curve, which can be cured somewhat by reducing the stepsize $\Delta$. But it is never able to achieve the same accuracy as an higher order integration schemes such as the DOP853 Runge-Kutta scheme of order $8^{th}$ with adaptive step size control [3], as demonstrated in Fig. 7. In theory, all line and curve quantities are supposed to be independent of the chosen method. In practice, they will differ.

**Interpolation Scheme** With vector field data given at an equidistant spatial sampling ("uniform grid") it is required to interpolate grid points to a smooth location. With linear interpolation the discretized manifold is $\mathscr{C}^1$, first order derivatives become discontinuous which shows up visibly in the curvature (Fig. 8(c)). Cubic interpolation via Catmull-Rom splines yields a somewhat smoother curvature, Fig. 8(b), yet artifacts are still visible. While Euler integration yields inaccurate results,



(a) analytic (b) bi/tri-cubic (c) bi/tri-linear

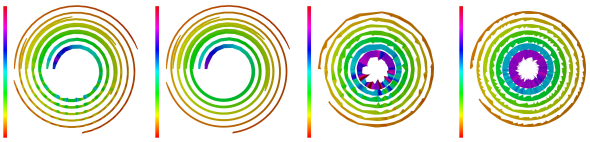


(d) analytic (e) bi/tri-cubic (f) bi/tri-linear

Figure 8: Curvature on Euler (upper row) and DOP853 integration (lower row).

the DOP853 integrator exhibits a remarkable behavior when visualization curvature (Fig. 8): it apparently approaches the curve by a combination of "undershooting" and "overshooting", which is more sensitive to the interpolation method.

**Discretization Resolution** The sampling density of an analytic function influences the accuracy of an integration scheme. As depicted in Fig. 9, increasing the grid resolution does smooth out the curvature as computed by the Euler scheme. For the more accurate DOP853 scheme however the "meandering" behavior as observed in Fig. 8 remains, just on a smaller scale.



(a) Euler, $16^3$ (b) Euler, $64^3$ (c) DOP, $16^3$ (d) DOP, $64^3$

Figure 9: Dependency of Frenet Ribbons and curvature on grid resolution.

**Differentiation Scheme** We implemented curvature computation directly by means of the definition (20) $\kappa = |\dot{t}|$. Given a discrete set of $N$ vertices along a line with $p_i \in M$, $v_i \in T_{p_i}(M)$ for $i \in [0,N)$, we arrive at

$$
\begin{aligned}
t_- &:= v_{i-1}/|v_{i-1}| & \Delta_- &:= |p_i - p_{i-1}| \\
t_0 &:= v_i/|v_i| & & \\
t_+ &:= v_{i+1}/|v_{i+1}| & \Delta_+ &:= |p_{i+1} - p_i| \\
\dot{t}_- &:= \frac{t_0 - t_-}{\Delta_-} & \dot{t}_+ &:= \frac{t_+ - t_0}{\Delta_+} \\
\dot{t} &= \frac{1}{2}(\dot{t}_- + \dot{t}_+) \quad \Rightarrow \quad \kappa = |\dot{t}|
\end{aligned}
$$

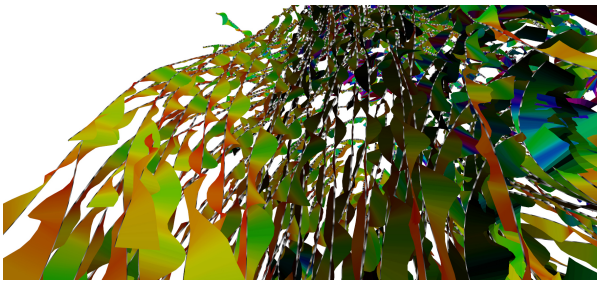
This is a second order scheme for differentiation, where division by the arc length $\Delta$ of a finite curve segment yields $\dot{t}$ (not $t'$), taking into account non-equidistant step sizes between succeeding points on the line. In the special case of constant step size, such as with Euler integration, the term $t_0$ will be discarded. When using the adaptive DOP853 scheme this symmetric formula using the tangential vector at $i-1$ and $i+1$ is important to yield smooth results. Boundary conditions ($i = 0$ and $i = N-1$) need to be treated differently. The computation of the torsion trivector is a direct implementation of eqn. (30), requiring one more numerical differentiation of $t'$:

$$
\ddot{t} := \frac{\dot{t}_+ - \dot{t}_-}{\Delta_+ + \Delta_-} \quad \Rightarrow \quad \tau = \frac{t_0 \wedge \dot{t} \wedge \ddot{t}}{\kappa^2} \tag{37}
$$

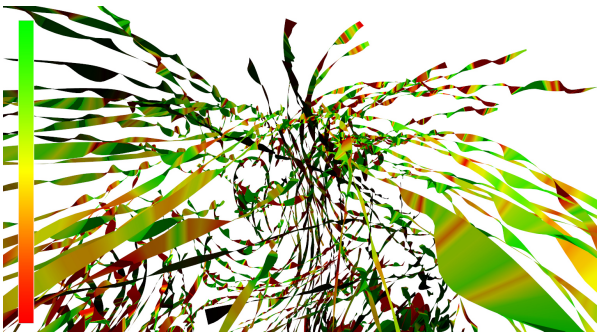
The method applied to a numerical dataset stemming from a large-scale computational fluid dynamic simulation is demonstrated in Fig. 10, showing Frenet Ribbons exposing the curvature bivector and torsion.

Curvature and torsion of pathlines are suitable indicators of the mixing quality of fluids [1]. Within a large

numerical dataset their depiction via Frenet Ribbons is useful for data mining purposes as slight deviations in curvature and torsion show up prominently.



(a) Frenet Ribbons with Color-Encoded Curvature



(b) Frenet Ribbons with Color-Encoded Torsion

Figure 10: Frenet Ribbons in a numerical vector field. Ribbons color-encoded by curvature or torsion, lines by proper time with gradient map.

## 4 CONCLUSION

In this article we have reviewed the Frenet-Serret equations describing the Differential Geometry of curves in the formalism of Geometric Algebra. This leads to a more intuitive formulation of curvature as a bivector and torsion as a trivector, explaining sign changes under reflection. The formalism is valid also in higher dimensional spaces, thereby generalizing vector algebra employing cross-products and quaternion formulations. The apparatus is applied to the numerical computation of integral curves in discretized vector fields and investigated for its dependency on numerical artifact such as interpolation scheme, integration method, sampling resolution and differentiation scheme. A real-world example is demonstrated on behalf of a dataset from computational fluid dynamics where Frenet Ribbons visualize the trajectories of test particles, exhibiting curvature and torsion.

## 5 ACKNOWLEDGMENTS

This research employed resources of the Center for Computation & Technology at Louisiana State University, which is supported by funding from the Louisiana legislature's Information Technology Initiative. Portions of this work were supported by NSF/EPSCoR Award No. EPS-0701491 (CyberTools). We thank Peter Wagner, Gunther Bergauer and our anonymous reviewers for comments and corrections.

## REFERENCES

[1] B. Bohara, W. Benger, M. Ritter, S. Roy, N. Brener, and S. Acharya. Time-curvature and time-torsion of virtual bubbles as fluid mixing indicators. *IADIS Computer Graphics, Visualization, Computer Vision and Image Processing 2010 (CGVCVIP 2010)*, 2010.

[2] C. J. L. Doran and A. N. Lasenby. *Geometric algebra for physicists*. Cambridge University Press, 2003.

[3] S. N. E. Hairer and G. Wanner. *Solving ordinary differential equations I, nonstiff problems, 2nd edition*. Springer Series in Computational Mathematics, Springer-Verlag, 1993.

[4] N. Fujimatsu and K. Suzuki. New interpolation technique for the cip method on curvilinear coordinates. *Journal of Computational Physics*, 229(16):5573 – 5596, 2010.

[5] A. J. Hanson. Quaternion Frenet Frames: Making Optimal Tubes and Ribbons from Curves. Technical report, Indiana University, 1994.

[6] A. J. Hanson and H. Ma. Visualizing flow with quaternion frames. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 108–115, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[7] W. Kühnel. *Differentialgeometrie*. Vieweg+Teubner, 2010.

[8] J. Sethian. Level set methods: An act of violence - evolving interfaces in geometry, fluid mechanics, computer vision and materials sciences. *American Scientist*, 1996.

[9] D. Stalling. *Fast Texture-Based Algorithms for Vector Field Visualization*. PhD thesis, Free University Berlin, 1998.

[10] S. Ushijima, I. Nezu, M. Sanjou, and Y. Sakane. Quintic spline interpolation (qsi) scheme with collocated grid on general curvilinear coordinates. XXIX IAHR CONGRESS, Beijing, China., Sept. 2001.

[11] T. Weinkauf and H. Theisel. Curvature measures of 3d vector fields and their applications. In *Journal of WSCG 2002, International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Plzen, Czech Republic, Number 10*, pages 507–514. WSCG, 2002.

[12] H. Weyl. *Philosophy of Mathematics and Natural Science*. Princeton University, Princeton, NJ, 1949.

# Automatic texture classification of metallographic images by Gabor Filter

Petr Kotas
Dept. of Applied Mathematics,
VŠB - Technical University of Ostrava,
17. listopadu 15, CZ-708 33 Ostrava, Czech Republic
Petr.Kotas@vsb.cz

Pavel Praks
Dept. of Applied Mathematics,
VŠB - Technical University of Ostrava,
17. listopadu 15, CZ-708 33 Ostrava, Czech Republic
Pavel.Praks@vsb.cz

Ladislav Válek
Research – Production Technology, ArcelorMittal Ostrava a.s., Ostrava, Czech Republic
ladislav.valek@arcelormittal.com

## ABSTRACT

In this paper an alternative method for the automatic pattern classification of metallographic images is presented. The aim of the pattern classification is to help monitoring the process quality in the steel plant of the company ArcellorMittal Ostrava plc, Ostrava, the Czech Republic. The here presented approach is based on the well known Gabor filter, which provides suitable results in various texture analysis applications. In our case, the real metallographic samples are firstly separated from the image background. Then, a texture extraction is provided. The extracted samples are processed by applying the Gabor filter with various properties, from which selected texture features are formed. Effects of a dimension reduction technique for quality of similarity retrieval are studied.

## Keywords

metallography, LSI, Gabor wavelet, texture analysis, industrial applications, image retrieval

## 1. INTRODUCTION

The aim of our research activity is to develop and test methods for visual analysis of digital images of metallographic samples. For quality modeling of metallographic images, it is important to retrieve visually similar images according to a user defined image, a query image.

The steel plant produces variety of steels with various properties, according to needs of consumers (pipes, building structures, energetics - metals for transformers, etc.). Steel samples from the cast billets are taken from continuous casting machine. These are crosscuts of the cast billets. These samples are conveyed into the metallography laboratory where they are mechanically adjusted. In order to stress a sample macrostructure, crosscut etching is done [Zeljkovic09].



Consequently, digital photographs of these etched crosscuts are being taken. In our digital image database, we try to recognize two basic production technologies of steels: alloyed and carbonic, see their different textures (Figure 2). The alloyed structure of image samples is characterized by visible segments. On the other hand, carbonic samples have small-grained structure without grains. Moreover, there are two basic shapes of metal samples, square and round.

Metallographic laboratories of ArcelorMittal Ostrava a.s. uses a system of standards for marking the macro structures. Usually this classification is not easy. For this reason, we represent digital images by feature vectors, which represent information hidden in textures of digital images of metallographic samples. Recently, we experimented with feature vectors obtained by the wavelet transformation and the eigen-space analysis. In all cases, the feature vector can be viewed as a sequence of image descriptors [Praks08a, Praks08b]. In this paper, an industrial application of the texture retrieval by the Gabor filter is presented.

The paper is structured as follows. Section 2 describes the theoretical background of the Gabor filter for the texture analysis. In Section 3, three used

image similarity models are presented. The experimental comparison of these approaches for the image classification follows in Section 4, while Section 5 closes the paper with conclusions and future works.

## 2. GABOR FILTER

In this section we describe fundamentals of 2D Gabor filters for the texture analysis. We show how texture samples could be represented in order to reduce the amount of data generated for the comparison. We refer to [Kruz00a], [Zhan00a] and [Man00a] for the general description of Gabor wavelets and further details in this topic. Also we refer readers interested in wavelets to [Fraz00a].

### Gabor wavelet

For an image *I(x,y)* with size *PxQ* is the Gabor wavelet transform defined as

$$\hat{I} = \iint_{S,T} I(x-s, x-t) * \bar{\Psi}_{mn}(s,t)\, ds\, dt, \qquad (1)$$

where *S* and *T* correspond to the wavelet size. Moreover, $\bar{\Psi}_{mn}(x,y)$ is the complex conjugate of $\Psi_{mn}(x,y)$, which is the self-similar Gabor wavelet. Finally, $\Psi_{mn}(x,y)$ is generated from the mother-wavelet

$$\Psi(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{\frac{-1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} \cos(2\pi x\lambda), \qquad (2)$$

where $\sigma_x$ and $\sigma_y$ are standard deviations of the gaussian envelope and $\lambda$ is the wavelength of the cosine factor.

In the next text *m* and *n* will range from 1 to *M* and from 1 to *N,* respectively.

The self-similar Gabor wavelet is defined through multiples of mother-wavelet:

$$\Psi_{mn}(x,y) = a^{-m}\Psi(\tilde{x},\tilde{y}) \qquad (3)$$

with *m* and *n* specifying scale and orientation of the function respectively, and

$$\begin{aligned} \tilde{x} &= a^{-m}(x\cos(\phi) + y\sin(\phi)) \\ \tilde{y} &= a^{-m}(-x\sin(\phi) + y\cos(\phi)) \end{aligned}, \qquad (4)$$

where $a>0$ and $\phi = n\pi/N$.

All variables above are adopted from [Zhan00a] and are defined as follows:

$$a = \left(\frac{U_h}{U_l}\right)^{\frac{1}{M-1}}, \qquad (5)$$

$$\lambda_{m,n} = a^m U_l, \qquad (6)$$

$$\sigma_{x,m,n} = \frac{(a+1)\sqrt{2\ln(2)}}{2\pi a^m (a-1) U_l}, \qquad (7)$$

$$\sigma_{y,m,n} = \frac{1}{2\pi\tan\left(\frac{\pi}{2N}\right)\sqrt{\frac{U_h^2}{2\ln(2)} - \left(\frac{1}{2\pi\sigma_{x,m,n}}\right)^2}}. \qquad (8)$$

We chose $U_h = 0{,}4$ and $U_l = 0{,}05$, because these values are widely used in the literature.

On Figure 2 a sample filter bank, we used in our experiments, with four angles and five frequency scales is shown.

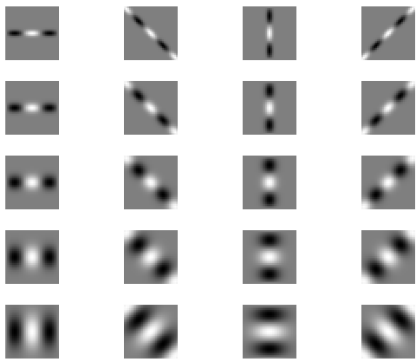


*Figure 1: Sample Gabor filter bank for M=5 and N=4.*

### Texture sample representation

Here we show how to use the frequency information obtained via Gabor wavelets to form features that fully represents the given texture sample.

$$\mu_{mn} = \iint_{xy} |\hat{I}(x,y)|\, dx\, dy \qquad (9)$$

$$\sigma_{mn} = \iint_{xy} \sqrt{\left(|\hat{I}(x,y)| - \mu_{mn}\right)^2}\, dx\, dy \qquad (10)$$

where symbols $\mu_{mn}$ and $\sigma_{mn}$ denotes mean and standard deviation of frequency responses to Gabor filters, respectively.

For the further refinement of the given metallographic samples we used a circular ratio, as we have to split square samples and circular ones. The circular ratio is defined as

$$C = \frac{A_R}{A_B}, \qquad (11)$$

where $A_R$ is area of the whole (segmented) sample and $A_B$ is area within the bounding box, respectively. The value of $C$ will be close to 1 for rectangular shapes and less than 1 for any other shapes. Since we have only rectangular and circular samples, we use only one feature for the shape description of the analyzed sample.

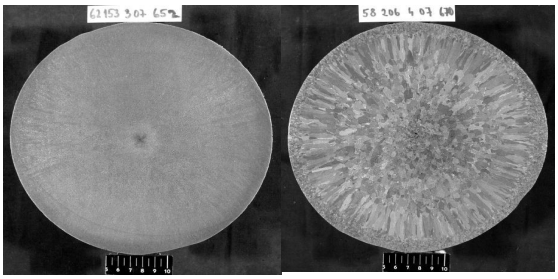


*Figure 2: Example of two different textures of metallographic samples: carbonic (left) and alloyed (right).*

## 3. SIMILARITY MEASURING

The goal of the similarity measurement is to find similar samples, which are described by the feature vector defined in Section 2.

For similarity measurement three types of measure were studied. We used a standard cosine similarity defined as

$$\varphi = \frac{x \cdot y}{\|x\|\|y\|}, \qquad (12)$$

which describes an angle between vectors $x$ and $y$. A small angle mean vectors are close ("similar") to each other.

The distance between two responses to Gabor filter is defined as

$$d_{mn}(i,j) = \sqrt{(\mu^i_{mn} - \mu^j_{mn})^2 + (\sigma^i_{mn} - \sigma^j_{mn})^2}. \qquad (13)$$

The distance between two feature vectors is defined as a sum of all responses to the Gabor filter,

$$D(i,j) = \sum_m \sum_n d_{mn}(i,j). \qquad (14)$$

We also studied an effect of the dimension reduction on accuracy of the measurement.

For this task the Singular value decomposition (SVD) was used. SVD is defined as

$$A = U \Sigma V^T \qquad (15)$$

where $A$ is a document matrix, $U$ and $V$ are matrices having reduced document and keyword dimension respectively and $\Sigma$ is a diagonal matrix with singular values. This kind of dimension reduction is also used for the latent semantic indexing (LSI). Details of LSI and SVD could be found in [Lars00a].

A dimension reduction has several positive affects on the original document matrix. It reduces amount of information needed for image descriptors. Moreover, it is used for automated noise reduction [Praks08b].

## 4. RESULTS

For our experiments we build two training databases both having unique texture samples that represents typical members of all metallographic images in our data set. These two databases have 3 and 5 distinct textures for each type of data.

For experiments we compared results of all three similarity measurement methods described in the previous section.

The query database was constructed from 42 different samples. Some samples were chosen to be the same as in the training database for verification.

Results for both training databases are shown in Table 1. For example, when the cosine similarity is used, there was only one image retrieval failure in 42 retrieved cases, which gives us the probability of incorrect retrieval 1/42 ~ 2.4 %. Some visual results are presented in Figure 3 and Table 2. In order to achieve well arranged results, only the most significant images are presented. Image retrieval results are presented by decreasing order of similarity. The query image is situated in the upper left corner. The similarity of the query image and the retrieved image is also presented. In order to achieve well arranged results, only 7 most visually similar images are presented.

| training set | lsi | cosine | tex. dist. |
|---|---|---|---|
| **3** | 28,5% | 2,4% | 31% |
| **5** | 12% | 2,4% | 35,7% |

*Table 1: Results for similarity measure – probability of incorrect retrieval.*

Our experiments show that all three methods prove to be successful in detecting similar samples, but each of them has advantages and disadvantages.

The cosine similarity is the most accurate and robust to training set size, but is sensitive to noise in the texture data.

The texture distance is relatively stable towards the size of the training data and seems to be less sensitive to noise.

LSI shows to have worse results for training set of size 3 and better size 5. This is not unexpected result. LSI in its nature reduces the number of required data needed for the object representation. This principle works well for large sets of data, but it is irrelevant for small data sets. This will result in removing information needed for the successful classification of the texture sample. For small data-sets, it is very difficult to distinguish the signal-to-noise ratio.

| Image | Similarity |
|---|---|
| SCK60U9.jpg | 1 |
| SCK60U14.jpg | 1 |
| SCK60U36.jpg | 1 |
| SDK53M29.jpg | 0.9998 |

*Table 2: An example of image retrieval results by the Gabor filter without LSI.*

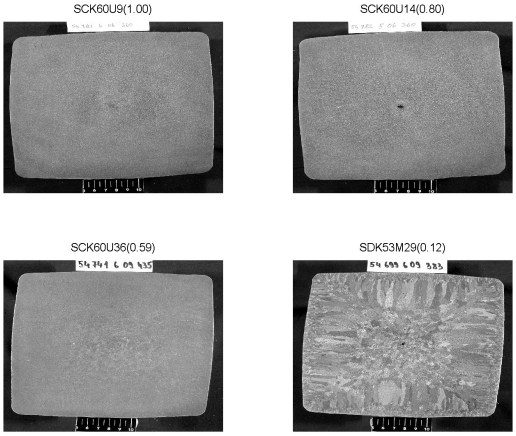



*Figure 3: An example of LSI image retrieval results by the Gabor filter.*

## 5. Conclusions

The objective of this research is visual monitoring of properties of various steels in the steel plant. The experimental digital images of metallography samples have been provided by the company ArcelorMittal Ostrava plc (Ostrava, Czech Republic).

In this paper, we presented an industrial application of the Gabor filter for texture retrieval of metallographic macro-structures. It is a combination of the Gabor filter representation with image retrieval by LSI, which was applied in a real industrial environment. The first results prove high performance of the image retrieval results. Our results indicate that the Gabor filter method can automatically recognize the shape (square vs. round) and the type of images found in our image database (alloyed vs. carbonic samples). The shapes of samples were recognized in all case without any problems. For the texture retrieval, the probability of the recognition error for alloyed vs. carbonic samples is only 2,4% for the cosine similarity measure. The here presented image retrieval results are very consistent with the human expert opinion [Praks08a]. In the future, it would be interesting to detect, extract and analyze detailed metallurgical relations in images, which are hidden in the digital image database of metallographic samples.

## ACKNOWLEDGMENTS

This work was supported by the project FR—TI1/432 of the Ministry of Industry and Trade of the Czech Republic.

## REFERENCES

[Fraz00a] Michael W. Frazier, An Introduction To Wavelets through linear algebra, Springer 1999

[Kruz00a] S.E. Grigorescu, N. Petkov and P. Kruizinga (2002) Comparison of texture features based on Gabor filters. IEEE Trans. on Image Processing, 11, 1160-1167

[Lars00a] L. Elden (2007) Matrix Methods in Data Mining and Pattern Recognition. SIAM

[Man00a] B. S. Manjunath, W. Y. Ma (1996) Texture Features for Browsing and Retrieval of Image Data. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18, 837-842

[Tai00a] T. S. Lee (1996) Image Representation Using 2D Gabor Wavelets. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18, 959-971

[Zhan00a] D. Zhang, A. Wong, M. Indrawan and G. Lu (2000) Content-based Image Retrieval Using Gabor Texture Features. IEEE Transactions PAMI., 13-15

[Praks08a] P. Praks, M. Grzegorzek, R. Moravec, L. Válek, and E. Izquierdo (2008): Wavelet and Eigen-Space Feature Extraction for Classification of Metallography Images. Information modelling and knowledge bases XIX. Vol. 166. pg. 190 – 199. ISBN 9781586038120

[Praks08b] Praks P., Kučera R., Izquierdo E.(2008): The sparse image representation for automated image retrieval. ICIP 2008. DOI:10.1109/ICIP.2008.4711682

[Zeljkovic09] Zeljkovic V., Praks P., Vincelette R., Tameze C., Valek L., Automatic Pattern Classification of Real Metallographic Images. 2009 IEEE Industry Applications Society Annual Meeting. Houston, TX

# Parallelization of a method for detecting non-stationary photometric perturbations in projection screens with CUDA

Antonio Díaz-Tula
Departamento de Ciencia de la Computación
Universidad de Oriente
Ave. Patricio Lumumba, 9500
Santiago de Cuba, Cuba
diaztula1@gmail.com

Miguel Castañeda-Garay
Departamento de Ciencia de la Computación
Universidad de Oriente
Ave. Patricio Lumumba, 9500
Santiago de Cuba, Cuba
mcgaray_cu@yahoo.es

Óscar Belmonte-Fernández
Departamento de Ingeniería y Cienca de los Computadores
Universitat Jaume I, Spain
Oscar.Belmonte@lsi.uji.es

## ABSTRACT

The human-computer interaction using large projection screens is gaining more space nowadays. For these screens several computer vision techniques have been developed that allow the user to interact with the system through the projected images using laser pointers, special pens and the hands. On this work is presented the parallelization of a method for the real-time detection of non-stationary photometric perturbations in projection screens using the Computed Unified Device Architecture, in order to overcome the elevated running time of the serialized implementation on CPU. A comparison of the results is presented to establish the acceleration of the parallel algorithm against its original version on CPU.

## Keywords

parallelization, photometric perturbation, projection screens, CUDA.

## 1. INTRODUCTION

High definition projection screens are gaining more space each day. Such screens are boosting the presence of multiple spectators, detailed model visualization, immersion sense and the creation of a natural environment of interactive collaboration between multiple users.

As a consequence, several computer vision techniques are being developed that allow users to interact with the system through projected images using laser pointers [Kirs98a], special pens [LaRo03a] and the hands [Koik01].

In [Mig09a] a local method for the real-time detection of non-stationary photometric perturbations in projected images was presented from modifications performed to the global method presented in [Jay04a] for the detection and removal of shadows in projected images.



The new method is based on computing the differences between the images of a projector frame buffer and the corresponding projected image captured by a camera. To carry out this comparison, a previous process of geometric and photometric calibration between the projector, the camera and the screen is needed.

To test this method a system prototype that uses a camera/projector pair was implemented, and it proved to be very robust when facing spatial variations of the projector's light intensity over the projection surface and the incidence on this surface of external locally-stationary factors.

But in the experimentation only about ten images per second were processed using a Core 2 Duo (2.66 GHz) processor, a NVIDIA GeForce 8400 GS GPU and a Logitech 9000 Pro Webcam, with a latency time of 95 milliseconds. This phenomenon cause visible differences of inaccuracy when detecting perturbations whose positions move across the screen [Mig09a].

Two main reasons were identified: the serial execution on CPU, and the latency time in the VRAM- to-RAM transfer of the projector frame buffer.

The method proposed in [Mig09a] is parallelizable as the information processing in several parts of the computation follows the Single Instruction Multiple Data (SIMD) model [Flynn72a].

The latency time is introduced during the copy of the projector frame buffer from VRAM to conventional RAM for the estimation process.

The Computed Unified Device Architecture (CUDA) gives the possibility to exploit the enormous parallel computing capabilities of the NVIDIA's Graphics Processing Unit (GPU), when applied to general purpose problems, as long as those problems are parallelizable by the SIMD model.

CUDA gives a subset of the C language to write kernels that run in parallel on GPU as a hierarchy of threads groups, with very low control and schedule overhead, fast barrier synchronization and shared memory usage [CUD09].

This makes CUDA a suitable technology to improve the running time of the method proposed in [Mig09a] ,because of two main reasons: it provides a powerful SIMD programming environment, and its global memory is located at VRAM as well as the projector frame buffer, that would decrease the latency time.

In this work is presented the **parallelization of the local method presented in** [Mig09a] **for the real-time detection of non-stationary photometric perturbations in projection screens using CUDA.**

The remainder of the content is structured as follows: section 2 gives a brief summary of the local method presented in [Mig09a] for the real-time detection of non-stationary photometric perturbations in projection screens; section 3 covers the parallelization of the previously mentioned method and finally, section 4 shows the results.

## 2. SUMMARY ON THE LOCAL METHOD FOR DETECTING PHOTOMETRIC PERTURBATIONS

The local method presented in [Mig09a] for the real-time detection of non-stationary photometric perturbations in projection screens is based on the comparison of two images: one of them represents the real image captured by the camera of the projected image (the camera image), and the other one is the image that "should" be captured by the camera (the estimated image).

Roughly speaking, if there were no lights or any other phenomena interfering with the projection process, the camera image and the estimated image should be very similar. Otherwise, these images must present differences in the regions where such perturbations are influenced.

To make a correspondence between the coordinates of the camera image, the projection surface image and projector frame buffer image, a geometric calibration process is needed, thus obtaining transference functions for the coordinate systems of such images. For the geometric calibration, the planar homography method described in [Suk01a] was used.

In the method presented in [Mig09a] the projector frame buffer resolution is higher than the camera resolution, which implies that a set of closely located pixels at the projector frame buffer are captured by the camera as a single pixel.

For this reason, the projector frame buffer is conceived as a matrix of rectangular regions, whose intersection is null and whose joint is equal to the buffer.. This partition is performed in order to obtain a matrix with a row and column number equal to the camera resolution, which in turn is the given resolution to the estimated image.

For each region of the projector frame buffer there is a single corresponding point in the estimated image. In turn, for each point of the estimated image there is a single corresponding point in the camera image, but for a given point in the camera image there may exist 0, 1 or more points in the estimated image (see Fig. 1).

To homogenize the color range of both the camera image and the estimated image a process of photometric calibration is needed.

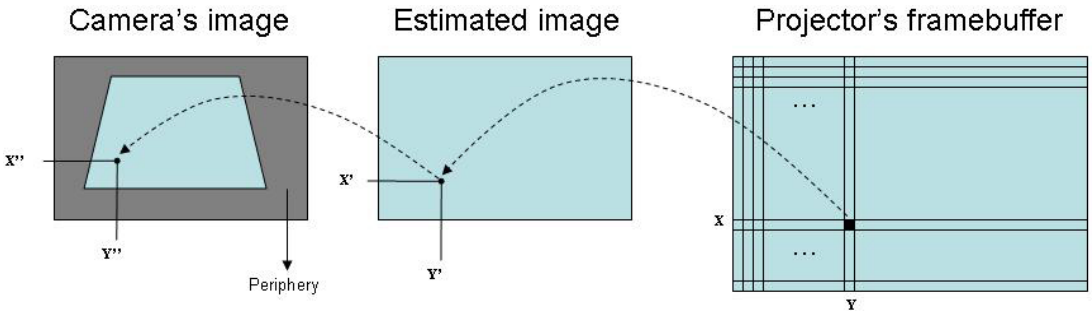



Figure 1 Correspondence between a region in the projector frame buffer, a point in the estimated image and a point in the camera image.

The photometric calibration process is established to make a correspondence between the color range of the camera image, the projection surface image and the projector frame buffer image.

This calibration is needed because the projector frame buffer image and the camera-captured image have different photometric ranges due to several internal and external factors such as: differences in the color spaces and brightness level between the projector and the camera; the location of the projector with respect to the camera, which cause brightness variations according to its position; the influence of the environmental light over the screen; the camera's internal features, as well as the adjustment of its intrinsic and extrinsic parameters; the existence of spots or irregularities on the projection surface, among others [Mig09a].

To perform this calibration, a model that produces transference functions between color spaces was used; those functions allow estimating the image that the camera should grab from the image of the frame buffer. For each region in the frame buffer such functions are obtained, one for each RBG color component. It differs from the method presented in [Jay04a] where the transference functions are obtained for the entire screen (global) and not for each region.

The color transference functions are previously evaluated for every possible value of each color component for each region of the frame buffer, and the results are stored in three-dimensional tables in order to avoid revaluating these functions each time during the estimation process.

Thus, the photometric calibration of each region in the frame buffer is given by three three-dimensional tables, one for each color component:

*byte [camHeight][camWidth][64] redTable*

*byte [camHeight][camWidth][64] greenTable*

*byte [camHeight][camWidth][64] blueTable*

To obtain the color components for a pixel in the estimated image, the average of the estimated color components for all the pixels in the frame buffer that belong to the corresponding region for the estimated pixel is computed.

Given a pixel in the frame buffer with coordinates *(x, y)* and color components *red*, *green* and *blue*, its corresponding coordinates in the estimated image are *(regX(x), regY(y))* and the estimated values are:

*estRed = redTable [regX(x)] [regY(y)][red /4]*

*estGreen=greenTable[regX(x)][regY(y)][green/4]*

*estBlue=blueTable [regX(x)] [regY(y)][blue /4]*

The function *regX(x)* gives the row of the estimated image that correspond to the pixel with row *x* in the frame buffer; the function *regY(y)* gives the column of the estimated image that correspond to the pixel with column *y* in the frame buffer.

The value of each color component is divided by 4 in order to reduce the amount of memory needed to store the tables of each region of the frame buffer.

The sequence of steps to detect the photometric perturbations is stated as follows:

1. Obtain the image of the projector frame buffer (in RGB format) in the variable *frameBuffer*.
2. For each pixel in this image with coordinates *(x, y)* obtain the coordinates *(x', y')* where *x'=regX(x)* y *y'=regY(y)*, then for the color components *red*, *green* and *blue* of *frameBuffer[x][y]* compute the estimated values by querying the entry *colorTable[x'][y'][color/4]*. Given that for several pixels in *frameBuffer* the same coordinates *(x', y')* will be obtained (for all that belong to the same region), the average must be computed for each color component.
3. Compare each pixel in the estimated image with its corresponding pixel in the camera image for each color component; if the difference between two values is greater than a given threshold, then a possible photometric perturbation may exist.

## 3. PARALLELIZATION OF THE LOCAL METHOD FOR DETECTING PHOTOMETRIC PERTURBATIONS

The first step is to copy the projector frame buffer to be processed with CUDA. One objective is to avoid the transference of this buffer to the RAM.

### Reading the projector frame buffer for its processing with CUDA

It can be consulted in various CUDA SDK examples, that it is possible to write parallel algorithms to post-process the frame buffer of a window through the CUDA's interoperability with OpenGL. As described in [CUD09], OpenGL buffer objects can be mapped with CUDA to be accessed from the kernels.

In our problem we need to read the entire projector frame buffer, that is, the "Desktop". Whereas OpenGL does not provide any functions to create rendering contexts, this must be done using the underlying operating system's API (hence, loosing portability that way). The method was implemented for the Microsoft Windows XP operating system.

To carry out the reading of the projector frame buffer we follow these steps:

- Create a top-level, layered window that covers the entire Windows's desktop; this window will be invisible.
- Create a hardware-accelerated OpenGL rendering context associated with this window and make it current.
- Create a Pixel Buffer Object (PBO) with enough memory to store the entire projector frame buffer (this is related to the screen size and color depth). The PBO memory is usually allocated in VRAM and controlled by OpenGL.
- Use the OpenGL's function *glBindBuffer* to link our PBO to the reading operations over the frame buffer.
- Use the OpenGL's function *glReadPixels* to perform the copy of the frame buffer to the PBO.

Then we just need to use the CUDA's API function *cudaGLMapBufferObject* to map the PBO and its content is ready to be accesed from CUDA's threads.

## Parallelization of the estimation and compare algorithms

### *3.2.1 Estimation algorithm in CPU.*

The estimation algorithm takes as input the projector frame buffer, the color tables of each region in this buffer, the dimensions of the estimated image and the projector frame buffer, and returns the estimated image, that is, the image that the camera should capture at exactly that moment.

The estimation algorithm in CPU is as follows:

Remark: All the coordinates are row major order.

Input: Projector frame buffer, dimensions of the frame buffer and the estimated image, color tables of all the regions in the projector frame buffer.

Output: Estimated image.

Step 1. Initialize the estimated image with 0.

Step 2. For each pixel in *frameBuffer* with coordinates *(i, j)* do steps 3 to 7:

Step 3. Compute the region to which belongs the pixel *(i, j)* in the estimated image:

*i'=i*cammeraHeight /frameBufferHeight*

*j'=j*cammeraWidth /frameBufferWidth*

Step 4. Obtain the pixel's color components:

*red = frameBuffer[i][j] & 0x000000FF*

*green=(frameBuffer[i][j] & 0x0000FF00)>>8*

*blue =(frameBuffer[i][j] & 0x00FF0000)>>16*

Step 5. Compute the estimated color components:

*estRed = redTable [i'][j'][ red /4]*

*estGreen = greenTable[i'][j'][green /4]*

*estBlue = blueTable [i'][j'][ blue /4]*

Step 6. Add the estimated values to the corresponding pixel in the estimated image:

*redEstimImg [i'][j'] += red*

*greenEstimImg [i'][j'] += green*

*blueEstimImg [i'][j'] += blue*

Step 7. Increase the count of pixels from *frameBuffer* that belong to the computed region:

*regionCount[i'][j']++*

Step 8. For each pixel in the estimated image with coordinates *(i',j')* divide by the pixel count for each color component:

*redEstim [i'][j'] /= regionCount[i'][j']*

*greenEstim[i'][j'] /= regionCount[i'][j']*

*blueEstim [i'][j'] /= regionCount[i'][j']*

Algorithm 1. Estimation algorithm in CPU.

### *3.2.2 Decomposition of the algorithm.*

As can be easily seen, the same steps repeat over different data, this allows a data parallelism over a shared address space [Gra03a].

We used the output data decomposition technique, where each output element can be independently computed as a function of the input. The value of each pixel in the estimated image (output) depends only on the corresponding frame buffer's region and its color tables (input).

This partition leads to the definition of a task as computing a pixel in the estimated image. The number of tasks is equal to the product of the estimated image's width and height. For example, 76800 tasks are obtained from a resolution of 320x240. This decomposition can be classified as fine texture according to its granularity. Figure 2 gives us a graphical scheme of the parallel algorithm.

Still an issue must be analyzed: all the regions of the frame buffer do not have the same size, i.e., the rows and columns number may be different for two or more regions. This may influence in the performance of the algorithm when the threads of the same warp diverge in their execution paths.

But there are several reasons in favor of this approach:

(i) There is uniformity in the sense that each thread computes exactly a pixel of the estimated image, thus being unnecessary any communication and synchronization mechanisms between threads.

(ii) Each thread will write on a single pixel in the estimated image, so there is no need to use atomic instructions (available only for computing capabilities 1.1 or above).

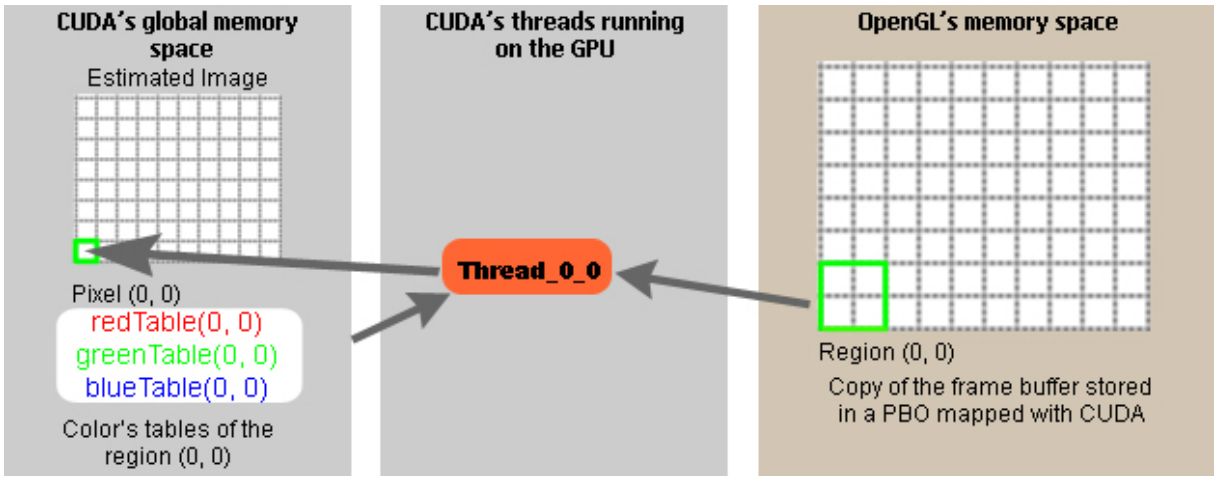



Figure 2. Scheme of the estimation algorithm if GPU.

### *3.2.3 Determining the execution configuration.*

When data decomposition is applied to a problem, generally the task mapping is static [Gra03a], where the tasks are distributed among the available processors before the algorithm execution.

Our problem adjusts to a Blocks Distribution Scheme [Gra03a], where the resulting matrix (the estimated image) is divided in areas of $k_1$ columns and $k_2$ rows, so that each thread must compute all the estimated pixels of a given area.

CUDA's threads are organized in a hierarchy of one-dimensional, bi-dimensional or three-dimensional thread blocks; those in turn are organized in a one-dimensional or bi-dimensional grid of blocks.

The GPU have a number of multiprocessors, and each multiprocessor have eight streaming processors. When the CPU launches the execution of a grid, its blocks are enumerated and distributed to the available multiprocessors. When a multiprocessor finishes the execution of a block, it gets assigned another non-executed block. This execution model is scalable, so we can define any number of threads without worrying about the number of multiprocessors.

When the number of thread blocks increases to a large amount, GPUs with a few multiprocessors will not be favored, as plenty of time will be used in distributing the non-executed blocks to the multiprocessors as they become available, and this time may be significant against the running time of each block.

For that reason the value of $k_1$ and $k_2$ must be obtained so each thread computes an area of $k_1$ x $k_2$ pixels of the estimated image. These values can be adjusted depending on the number of available multiprocessors, thus giving more scalability to the implementation.

The thread blocks are set to be bi-dimensional and have 16x16 = 256 threads, a value that is recommended in [CUD09] to obtain a good performance. The blocks grid is also bi-dimensional and it size will be:

$grid.x = ceil( cameraWidth / 16 * k_1 )$

$grid.y = ceil(cameraHeight / 16 * k_2 )$

For instance, if we want a maximum number of 512 blocks per multiprocessor we proceed as follows:

$$ceil(\frac{cameraWidth}{16 * k_1}) \geq \frac{cameraWidth}{16 * k_1}$$

$$ceil(\frac{cameraHeight}{16 * k_2}) \geq \frac{cameraHeight}{16 * k_2}$$

$$\frac{cameraWidth * cameraHeight}{256 * k_1 * k_2 * mpCount} \leq 512$$

$$k_2 \geq \frac{cameraWidth * cameraHeight}{131072 * k_1 * mpCount}$$

It is desirable that $k_1$ be a divisor of *cameraWidth* and $k_2$ be a divisor of *cameraHeight*, because, then the number of pixels to estimate is uniformly distributed among all threads.

Table 1 shows some possible execution configurations.

| Camera resolution | Multiprocessors (MPs) count | $k_1$ | $k_2$ | Blocks per MPs |
|---|---|---|---|---|
| 320x240 | 2 | 1 | 1 | 150 |
| 800x600 | 2 | 1 | 2 | 469 |
| 1024x768 | 2 | 1 | 3 | 512 |

Table 1. Execution configurations for different camera resolution and multiprocessors count.

### *3.2.4 Parallel estimation and compare algorithms.*

We expose next the parallel estimation algorithm:

Remark: All the coordinates are row major order.

Input: Initial coordinates of the estimated image area that the thread will compute, values of $k_1$ and $k_2$, color tables and projector frame buffer regions of the estimation area. Size of the projector frame buffer and the estimated image.

Output: Corresponding area of the estimated image.

Step 1. For each pixel in the estimated image area with coordinates *(i', j')* do steps 2 to 8:

Step 2. Initialize *redSum*, *greenSum* and *blueSum* with zero.

Step 3. Compute in *pixelCount* the number of pixels in the corresponding projector frame buffer region.

Step 4. For each pixel in the corresponding *frameBuffer* region with coordinates *(i, j)* do steps 5 to 7:

Step 5. Obtain the color components of the pixel from *frameBuffer*:

```
red   =  frameBuffer[i][j]& 0x000000FF
green =(frameBuffer[i][j]& 0x0000FF00)>>8
blue  =(frameBuffer[i][j] & 0x00FF0000)>>16
```

Step 6. Compute the estimated values for each color component:

```
estRed   =  redTable  [i'][j'][   red /4]
estGreen = greenTable[i'][j'][green /4]
estBlue  =  blueTable [i'][j'][  blue /4]
```

Step 7. Add the estimated values:

```
redSum   [i'][j'] += estRed
greenSum[i'][j'] += estGreen
blueSum  [i'][j'] += estBlue
```

Step 8. Write the averaged results in the estimated image:

```
redEstimImg   [i'][j']=  redSum/pixelCount
greenEstimImg[i'][j']=greenSum/pixelCount
blueEstimImg  [i'][j']=  blueSum/pixelCount
```

Algorithm 2. Parallel estimation algorithm in GPU.

The algorithm for comparing the images was also parallelized. The same execution configuration that was previously exposed is use for this algorithm.

The parallel version of the compare algorithm introduces some execution overhead, since the camera image must be copied to the CUDA's global memory to perform the comparison with the estimated image (that already is at global memory).

Next we expose the parallel compare algorithm:

Remark: All the coordinates are row major order.

Input: Initial coordinates of the estimated image area and the camera image area that the thread will compare, values of $k_1$ and $k_2$, size of the images.

Output: Binary matrix area with 1 in the pixels where a photometric perturbation may exists.

Step 1. For each pixel in the estimated image area and the camera area with coordinates *(i, j)* do steps 2 to 3:

Step 2. Compute the differences between color components:

```
red= abs(redEstimImg  [i][j] – redCamera  [i][j])
green = abs(greenEstimImg[i][j] – greenCamera[i][j])
blue=abs(blueEstimImg[i][j] – blueCamera [i][j])
```

Step 3. Compare and write the results in the output matrix:

```
outImg[i][j]=(red>redThreshold||
green>greenThreshold || blue>blueThreshold)?1:0
```

Algorithm 3. Parallel compare algorithm in GPU.

## Some implementation details.

### *3.3.1 Execution phases.*

The process of working with the GPU was divided in three main phases:

- Initialization phase: the necessary memory is allocated in CUDA and in the host, the color tables for each region are copied to CUDA's global memory, several constants are initialized and the execution configuration for the algorithms is determined.

- Execution phase: the parallel algorithms are invoked. The normal order should be:

  Estimation algorithm, requires no transference between the host and the device.

  Compare algorithm, requires two transferences between host and device: receives the camera image, and return the resulting binary matrix.

  There is some relaxation in the sense that both the estimation algorithm and the compare algorithm can be called more than once repeatedly, although if the compare algorithm is called before any call to the estimation algorithm then the results are inconsistent.

- Termination phase: all resources are released from the GPU and CPU, and the working session with CUDA is closed.

### *3.3.2 Use of shared memory.*

Global memory accesses are less time-expensive while less memory transactions are require. If all threads in a half-warp (0 to 15 or 16 to 31) follow a memory access pattern (which differs according to the computing capability) then the memory accesses can be **coalesced** and only a few (one or two) memory transactions are required, thus improving performance.

Due to the nature of our problem, when the threads of a half-warp accesses its color tables to estimate a color, say the red, each can have access to any of 64 bytes, so the total area they can address is 64*16=1024 bytes, and one of the requirements for coalescence is that all the threads in the half-warp accesses an aligned memory segment having at most 128 bytes [CUD09] for computing capabilities 1.2 or above. For computing capabilities 1.0 and 1.1 the coalescence requirement are stricter, so no coalescence will be met for any computing capabilities.

Two choices are available: the use texture fetches or shared memory; we chose the second one since it is possible to obtain a bank-conflicts free distribution with a probability of 1/16 that the desired value exists in shared memory for each thread and each color component.

As the blocks have 16x16 threads, we define a matrix of shared memory with the same size for each color component, which totalize (256 threads)*(4 bytes)*(3 colors) = 3 Kb of shared memory per block. This low value allows for several active blocks per multiprocessor, improving the overall performance.

Figure 3 gives an idea of the use of shared memory in the estimation algorithm.

This distribution is bank-conflicts free. Table 2 shows the distribution of each word of shared memory to the banks of shared memory.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| … | | | | | | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Table 2. Distribution of each word of shared memory in a 16x16 word matrix to the banks.

The number inside each cell represents the bank of shared memory on which the 32-bit word of shared memory is located. As can be seen, all threads of each half-warp own a word of shared memory that lies in a distinct bank, thus avoiding bank conflicts.

This shared memory size per thread gives the possibility to estimate up to 16 different values per color component without accessing the global memory, because 256/4 = 64 entries in the color table, divided by 4 bytes on each shared memory word = 16 values.

It should also be considered that it's not likely that there exist many sudden variations of color in an image, because generally the changes of colors are softened and progressive.

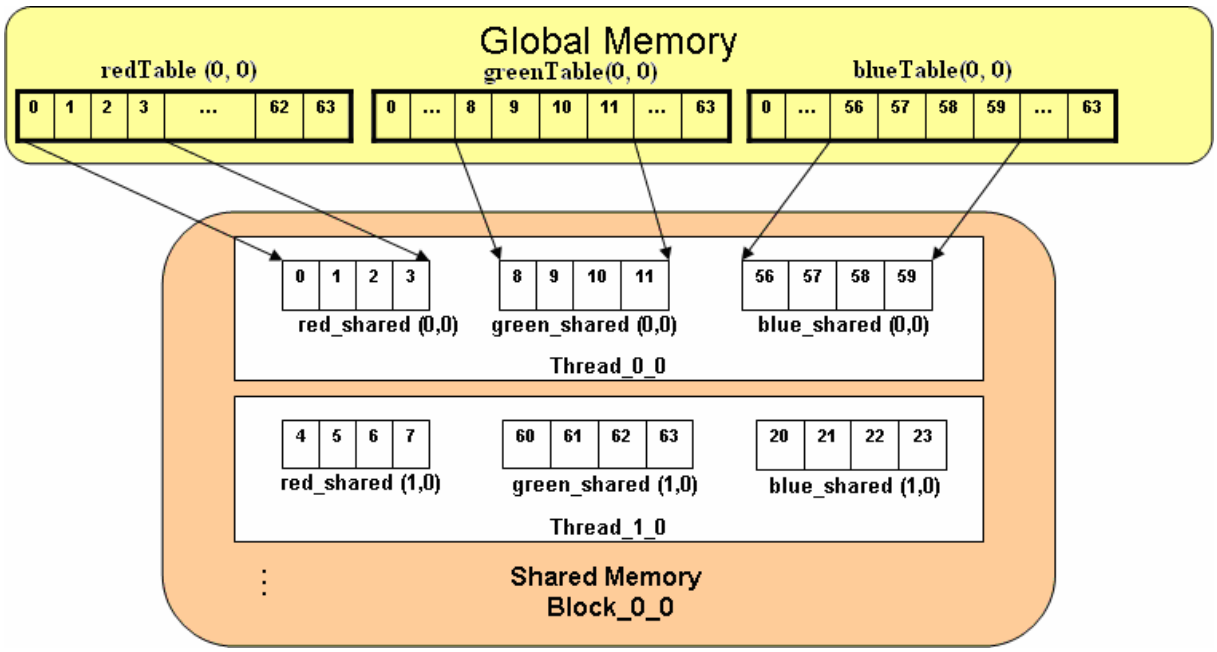



Figure 3. Use of shared memory.

## EXPERIMENTATION AND RESULTS

To evaluate the parallel implementation of the local method presented in [Mig09a] for the real-time detection of non-stationary photometric perturbations in projection screens some experimentation was made.

The parallel algorithms were integrated to an existing system prototype implemented in Java through the Java Native Interface (JNI). A little effort was needed to make the system call either, the existing serial implementation on CPU or the new parallel implementation with CUDA.

The experimentation was carried out in a Core 2 Duo processor at 2.66 MHz and a GeForce 8400 GS GPU (having two multiprocessors). The screen resolution was 1024x768 and the camera resolution was 320x240, both with 32 bit color in RGB format. The development environment for the experiment was NetBeans IDE 6.5 over Microsoft Windows XP SP2, and the version of CUDA 2.2.

Table 3 shows the results of running both the serial and parallel algorithms in the system prototype. For the parallel algorithms, experimentation was made both using shared memory and not using shared memory.

As can be seen, without using shared memory a speedup of 1.7x was achieved, in contrast with the higher **2.8x** speedup obtained when using shared memory.

| | Serial implementation | Parallel implementation | Parallel implementation (without using shared memory) |
|---|---|---|---|
| Latency time | 95 ms | 35 ms | 57 ms |
| Average FPS | 10 | 28 | 17 |

Table 3. Experimentation results.

The number of concurrent threads per iteration in the parallel implementation is equal to 320x240 = 76800 for the estimation algorithm, plus 76800 for the compare algorithm, making a total of 153600 threads.

## CONCLUSIONS

In this work, it was presented the parallelization of the local method presented in [Mig09a] for the real-time detection of non-stationary photometric perturbations in projection screens using the Computed Unified Device Architecture.

The implementation requires neither communication nor synchronization between threads. It is also designed to be scalable and compatible with computing capabilities 1.0 or above, and a bank conflicts free access to shared memory is used in order to improve performance, obtaining a speedup of 2.8x in the experimentation.

Still some other optimizations may be introduced to the implementation in the future for trying to achieve better results.

## REFERENCES

[CUD09] NVIDIA CUDA™ Programming Guide 2.2, 2009 NVIDIA Corporation.

[Flynn72a] Flynn, M. J.: Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, vol. C-21, Sept. 1972.

[Gra03a] Grama, A., Karypis G. et al., Introduction to parallel computing, Addison-Wesley, 2003.

[Jay04a] JAYNES C., WEBB S., STEELE M.: Camera-based detection and removal of shadows from interactive multiprojector displays. IEEE Transactions on Visualization and Computer Graphics 10, 3 (2004).

[Kirs98a] Kirstein, C., Muller, H.: Interaction with a projection screen using a camera-tracked laser pointer. In: Proceedings of the International Conference on Multimedia Modeling. IEEE Computer Society Press (1998).

[Koik01] Koike, H., Sato, Y., Kobayashi, Y.: Integrating paper and digital information on EnhancedDesk: a method for real-time finger tracking on augmented desk system. In: ACM Trans. On CHI, 8 (4), pp. 307--322 (2001).

[LaRo03a] La Rosa, F., Costanzo, C, Lannizzotto, G.: VisualPen: A Physical Interface for natural human-computer interaction. In: Physical Interaction (PI03) – Workshop on Real World User Interfaces. 2003.

[Mig09a] Castañeda-Garay, M., Belmonte-Fernández, O., Gil-Altaba, J., Pérez-Rosés, H., Un Método para la Detección en Tiempo Real de Perturbaciones Fotométricas en Imágenes Proyectadas, Congreso Español de Informática Gráfica CEIG'09, San Sebastian, Sept. 9-11 (2009), Páginas 239-242. The Eurographics Digital Library, http://diglib.eg.org.

[Suk01a] Sukthankar, R., Stockton, R.G., Mullin, M.D.: Smarter presentation: Exploiting homography in camera-projector systems. In: Proceedings of International Conference on Computer Vision, pp 247--253. Vancouver, Canada, July 9-12 (2001)

# Rendering Pipeline Modelled by Category Theory

Jiří Havel
Faculty of Information Technology
Brno University of Technology
ihavel@fit.vutbr.cz

Adam Herout
Faculty of Information Technology
Brno University of Technology
herout@fit.vutbr.cz

**ABSTRACT**

This paper describes basic concepts from category theory, which are commonly used in functional programming. These concepts are applied to shader programming and to the rendering pipeline and the whole rendering pipeline is formally modelled using category theory. This model can be used for more abstract and formal approach to shader programming. Mathematical formalization of the rendering pipeline and its stages can be helpful in shader compiler design, for proving algorithms, complexity analysis, and other tasks.

**Keywords:** Rendering, Shaders, Category Theory

## 1 INTRODUCTION

Category theory [Wal92, ML71] is an abstraction of mathematical structures and relations between them. It started as a "generic abstract nonsense", but now it is heavily used not only in mathematics, but also in computer science and especially functional programming. Many categorical concepts give rise to common pieces of code used to combine computations together. >From these abstractions, especially functors and monads are almost ubiquitous [Mog89].

Although the rendering pipeline is programmable, its overall structure is basically fixed. The design of programming languages for shader programming perfectly corresponds to the structure. Some experimental shading languages tried to offer a slightly more abstract way. Two notable examples are the Gpipe[1] library for Haskell and Sh [MQP02, MDTP+04] for C++. Both are, however, limited to OpenGL 2 or DirectX 8 functionality.

The Sh library views shaders as objects, that can be combined using two operations – serial and parallel composition. These two operations, however, perfectly correspond to a category with products, which will be described in the next section.

The GPipe library achives the same functionality by a different approach. It operates on streams of primitives and transforms them by series of functions – using functors from the category theory.



[1] http://www.haskell.org/haskellwiki/GPipe

The rest of the paper will discuss correspondences between those approaches. Section 2 will introduce a category-based model of the rendering pipeline stages and extend programming of pipeline stages to a description of the whole pipeline. Section 3 will raise the abstraction of the pipeline description to a composition of stream transformers. Section 4 summarizes the introduced categories as a model of the rendering pipeline.

## 2 CATEGORY WITH STREAMS

A category ($\mathbf{C}$) consists of a set of objects (obj $\mathbf{C}$) and a set of arrows or morphisms (arr $\mathbf{C}$) that link these objects. An arrow $f : A \to B$ has a domain $A$ and a codomain $B$ from obj $\mathbf{C}$. The set of arrows from $A$ to $B$ is denoted as $\mathrm{Hom}_{\mathbf{C}}(A,B)$. Arrows must be composable (1a), every object must have one identity arrow (1b) (1c) and arrow composition must be associative (1d).

$$\forall f : A \to B, g : B \to C \; \exists (g \circ f) : A \to C \tag{1a}$$

$$\forall O \in \mathrm{obj}\ \mathbf{C} \; \exists 1_O : O \to O \tag{1b}$$

$$\forall f : A \to B, 1_B \circ f = f \circ 1_A = f \tag{1c}$$

$$h \circ (g \circ f) = (h \circ g) \circ f \tag{1d}$$

A classical example is the category **Set**, whose objects are sets and arrows are mappings between those sets.

N-ary functions and tuples are modelled using products – composite objects. Product $P$ is an object consisting of objects $O_i, i = 0, \ldots N$ with projection arrows $(p_i : P \to O_i)$ that extract its member objects. For every object $Z$ that has arrows $a_i : Z \to O_i$ to all members of $P$, exactly one arrow $a_P$ to the product exists, such that $a_i = p_i \circ a_P$, see Figure 1. An exaple of the product type can be the structure data type, that is a product of its elements. When describing function, the arrow $a_P$ is a combination of functions returning the elements of a structure to a function returing the whole structure.

Let us define a category of GPU programs called **Gpu**. Objects of **Gpu** are basic and structured datatypes of graphical shaders and arrows are functions (both built-in and composite). Both structured

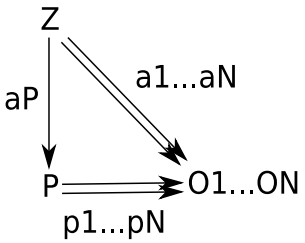



Figure 1: Product in a category

datatypes (structure and array) fullfil the requirement for product types[2] as do the basic vector types. With streams from the following subsection, every possible shader will be an arrow in **Gpu**.

## 2.1 Streams are Functors

Functor is a structure-preserving mapping between categories. Functor $F : \mathbf{A} \to \mathbf{B}$ between categories $\mathbf{A}$ and $\mathbf{B}$ consists of a mapping $F_{\text{obj}} : \text{obj}\,\mathbf{A} \to \text{obj}\,\mathbf{B}$ and mappingss $F_{A_1,A_2} : \text{Hom}_{\mathbf{A}}(A_1,A_2) \to \text{Hom}_{\mathbf{B}}(F_{\text{obj}}(A_1),F_{\text{obj}}(A_2))$ for every pair of objects $A_1$, $A_2$ from $\mathbf{A}$. Functors must also preserve arrow composition and identity arrows.

Homogenous abstract data types like lists, stacks, or queues are constructed by functors. The $F_{\text{obj}}$ creates structured types from the basic ones. The mappings for arrows convert simple arrows to arrows working with new types – usually doing the same for every element of the new type. In functional programming, this family of mappings is denoted by *map*. $map : (X \to Y) \to (F(X) \to F(Y))$, where $F$ is a functor, i.e., for a given arrow on the simple data type, *map* defines the corresponding arrow on the structured data type.

In shader programming, one abstract data type is ubiquitous – the stream of values (vertices, primitives, fragments). It is a sequence of elements, that is, however, never handled directly, but implicitly by the streaming nature of the rendering pipeline. Stream types in the category **Gpu** are constructed using a functor $Stream : \mathbf{Gpu} \to \mathbf{Gpu}$. This functor creates streams of elements of any basic type (and recursively streams of streams). For every $X$ and $Y$ from obj **Gpu**, the mapping for arrows is simply defined as

$$map : (X \to Y) \to (Stream(X) \to Stream(Y))$$
$$map(f)(x_1,\ldots,x_n) = (f(x_1),\ldots,f(x_n)).$$

The mapping $map(f)$ transforms every stream element by the function $f$, see Figure 2.

Streams are not the only functors in the category **Gpu**, but arrays and basic vector types are functors as well. For these data types, *map* works similarly to its stream counterpart.

[2] Objects like sums (discriminated unions) or exponentials (partially applied functions) are hard to express on GPU, so will not be used in this paper.

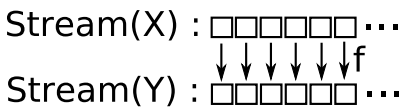



Figure 2: Functorial transformation principle.

For categories $\mathbf{A}$ and $\mathbf{B}$ a category $\mathbf{B}^{\mathbf{A}}$ exists: the *functor category*, whose objects are functors from $\mathbf{A}$ to $\mathbf{B}$. Arrows of this category are called *natural transformations*. For every functor $F$ and $G$ from $\mathbf{B}^{\mathbf{A}}$, natural transformation $\phi : F \to G$ and arrow $f \in A$, must $\phi \circ F(f) = G(f) \circ \phi$.

Natural transformations change only the structure of an abstract data type, but the elements of the type are left unchanged. For example, a transformation between an array of streams and a stream of arrays is a natural transformation. Natural transformations are heavily used in the following subsections.

## 2.2 Streams are Monads

Monad is a special type of functor used in functional programming to represent computations and control structures, to embed side effects, or model a processing pipeline.

Monads in category theory is a functor $F$, together with two natural transformations $\eta : 1_{\mathbf{C}} \to F$ and $\mu : F^2 \to F$ ($F^2 = F \circ F$). The corresponding triplet in functional programming consists of a functor $F$ and mappings $unit : A \to F(A)$ and $join : F(F(A)) \to F(A)$ [Mog88].

Mapping *unit* creates the monad type from one element and *join* merges two layers of the monad to one. In the category **Gpu**, the transformation $unit : X \to Stream(X)$ creates a stream with a single element. The transformation $join : Stream(Stream(X)) \to Stream(X)$ joins a stream of streams to one single stream by concatenation.

In functional programming, the *bind* transformation is used more than *join* and it better describes the properties of monads.

$$bind : (X \to F(Y)) \to (F(X) \to F(Y))$$
$$bind(f) = join \circ map(f)$$

In **Gpu**, the transformation $bind : (X \to Stream(Y)) \to (Stream(X) \to Stream(Y))$ uses the provided mapping to transform a stream. The type shows that every element of the input stream is used to create a new stream and such new streams are concatenated together. In other words, $0-N$ elements can be created from every single input element and the input elements are processed separately, as shown by Figure 3.

A perfect example of monadic processing of a stream is the geometry shader. From every primitive of the input stream, zero or more primitives are generated and those new streams are concatenated. Also, the fragment

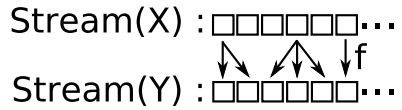



Figure 3: Monadic transformation schema.

shader can be described by a monad, as it can output empty streams or streams with a single element.

### 2.3 Streams are Comonads

Functor and monad are sufficient to describe stream transformers that access single elements of a stream. For accessing multiple elements, a categorical dual to a monad can be used – the comonad. Comonads can represent some information in a context. In the case of streams, the context of each element are the neighboring elements.

As dual functor to a monad, comonad is a functor with two natural transformations in opposite direction as the monad. These are $extract : F \to 1_{\mathbf{C}}$ and $duplicate : F \to F^2$. The functional programming forms are $extract : F(X) \to X$ and $duplicate : F(X) \to F(F(X))$. Mapping *extract* discards the context of a value and *duplicate* duplicates the context for every input.

Similarly to monad function *bind*, comonad has its dual *extend*.

$$extend : (F(X) \to Y) \to (F(X) \to F(Y))$$
$$extend(f) = map(f) \circ duplicate$$

As the type suggests, *extend* can not change the element count, but contrary to *bind*, the output elements depend on the context of the input elements as shown by Figure 4.

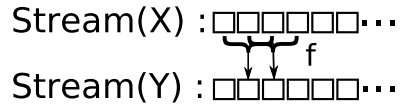



Figure 4: Comonadic transformation schema.

In **Gpu** several implementations of the comonad for streams are possible. Preceding or following elements can form a context of a stream element – the underlying implementation can be a delay link for example. The actual implementation is not important for the scope of this paper.

The primitive assembly can be viewed as a comonad (followed by a monad); however, because of the independence on actual stream contents, primitive assembly can be also modelled as a natural transformation. The tesselation control or hull shaders have also the comonadic structure, although they do not operate on stream but on array with index.

## 3 PIPELINE CATEGORY

The previously introduced category **Gpu** mixes the pipeline structure and the stages implementation. We can construct a category, that models only the pipeline structure and abstracts the actual implementation of the stages.

For a category **C** with monad $M$, which contains arrows of type $A \to M(B), A, B \in \text{obj}\,\mathbf{C}$, exists another category **K**,

$$\text{obj}\,\mathbf{K} = \text{obj}\,\mathbf{C}$$
$$\text{Hom}_{\mathbf{K}}(A,B) = \text{Hom}_{\mathbf{C}}(A, M(B))$$

The arrow composition in **K** is defined as $g_{\mathbf{K}} \circ f_{\mathbf{K}} = bind(g_{\mathbf{C}}) \circ f_{\mathbf{C}}$, so kategory **K** expresses composition of stream transformations. This category is called *Kleisli category* of **C**.

Dual to Kleisli category, also the *CoKleisli category* **L** for every category **C** exists, with a comonad $N$.

$$\text{obj}\,\mathbf{L} = \text{obj}\,\mathbf{C}$$
$$\text{Hom}_{\mathbf{L}}(A,B) = \text{Hom}_{\mathbf{L}}(N(A), B)$$

Similarly, the arrow composition in **L** is defined as $g_{\mathbf{L}} \circ f_{\mathbf{L}} = g_{\mathbf{C}} \circ extend(f_{\mathbf{C}})$.

For the category **Gpu**, we can construct a *pipeline* or *stream category* **Pipe**. Objects of this category are basic and structured shader types and arrows are functions of type $Stream(X) \to Stream(Y)$. Arrows fall to three groups.

- $map(f)$, $f$ is an arrow of **Gpu** without streams.
- $bind(g)$, $g$ is an arrow of the Kleisli category of **Gpu**.
- $extend(h)$, $h$ is an arrow of the CoKleisli category of **Gpu**.

This category provides a superset of all stream transformations possible on GPU. In functional programming, this kind of structure is called *Arrows* [Hug00, Pat01].

From the axioms for functor, monad and comonad [GLMP01], the following equivalences can be derived:

$$map(f) \circ map(g) = map(f \circ g) \quad (2a)$$
$$map(f) \circ bind(g) = bind(map(f) \circ g) \quad (2b)$$
$$bind(g) \circ map(f) = bind(g \circ f) \quad (2c)$$
$$map(f) \circ extend(h) = extend(f \circ h) \quad (2d)$$
$$extend(h) \circ map(f) = extend(g \circ map(f)) \quad (2e)$$
$$extend(h) \circ bind(g) = map(h) \circ duplicate \circ join \circ map(g) \quad (2f)$$
$$bind(g) \circ extend(h) = bind(g \circ h) \circ duplicate(x) \quad (2g)$$

When applied to the shader programming, these equations are intutive. Equation (2a) shows that two vertex-shader-like stages can be composed to one. Equations (2b) and (2c) show the composition of a geometry-shader-like stage with the following or preceding vertex shader. Equations (2d) and (2e) show the same for a comonadic shader.

The category **Gpu** has product types – tuples (structures) of basic types or streams. For arrows of type $map(f)$, the tuples of streams are isomorphic to streams of tuples. Therefore, also streams of tuples have the properties of product types. The language *Sh* uses these properties for parallel composition of shaders.

When arrows of type $bind(f)$ are considered, the product properties are lost. As outputs of two arrows can be differently structured, they cannot be generally merged together without mutual affection. This limits *Sh*-style parallel composition capabilities to vertex shader, tesselation evaluation, and fragment shader without discard.

The stream category can contain only streams of tuples. Because tuples in **Pipe** are not products, the arrows generally can not be combined in parallel. This parallel composition is limited to *functorial* and *comonadic* arrows.

# 4 MODEL OF THE RENDERING PIPELINE

Randering pipeline can be described using two categories. **Gpu** models complete GPU functionality from implementation of shader stages to the whole pipeline structure. Pipeline category **Pipe** models the pipeline structure by composition of simple shaders without considering their implementation.

The simple shaders can be classified according to their capabilities. The pipeline stages can be classified similarly.

- *Functorial* shaders change only single stream elements. The results do not depend on the context and the stage cannot change the stream's structure. Existing stages: Vertex shader, Tesselation evaluation shader, Fragment shader without discard.

- *Monadic* shaders can expand and remove elements. Their input is limited to one element. Existing stages : Rasterization, Fragment shader with discard, Fragment tests, Geometry shader, Hardware tesselator.

- *Comonadic* shaders can process the context of the element – the input consists of several elements (possibly the whole stream). The output is, however, limited to one element. Existing stages : Primitive assembly.

Using equivalences from (2), the stages can be composed from multiple simple shaders. Functorial stages can be composed only from functorial shaders. The (co)monadic stages can be composed from both (co)monadic and functorial shaders, as every (co)monad is a functor.

Also *functorial* and *comonadic* stages can be constructed using parallel composition. Both *Sh* and *GPipe* use only equivalence (2a). Following equivalences can be used to extend the model capability to cover geometry and tesselation shaders.

# 5 CONCLUSION

This paper introduced a model of the rendering pipeline using category theory. Although the mathematics in this paper is not novel, it is not commonly seen in the field of computer graphics. Two categories are defined and used: **Gpu** describing pipeline capabilities, structure and implementation and **Pipe** for abstacting the pipeline structure and composition from simple shaders.

The formalism introduced in this paper can be used for classification of different shader operations and for automatic optimization of shader programs on inter-stage level. Notably the equivalences from equation 2 form rewrite rules for moving computations between different stages of the rendering pipeline. However application of these rules is not trivial. Following work will focus on searching suitable rules, probably using genetical algorithms.

The model is inspired by two actual shader languages and will be used for their extension. The **Pipe** category can be also possibly used to describe generic stream processing.

# ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant FIT-10-S-2 and by the research project "Security-Oriented Research in Information Technology" CEZMSMT, MSM0021630528.

# REFERENCES

[1] Neil Ghani, Christoph Lüth, Federico De Marchi, and John Power. Algebras, coalgebras, monads and comonads, 2001.

[2] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37:67–111, May 2000.

[3] Saunders Mac Lane. *Categories for the Woring Mathematician*. Springer, 1971.

[4] Michael McCool, Stefanus Du Toit, Tiberiu Popa, Bryan Chan, and Kevin Moule. Shader algebra. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 787–795, New York, NY, USA, 2004. ACM.

[5] Michael D. McCool, Zheng Qin, and Tiberiu S. Popa. Shader metaprogramming. In *HWWS '02: Proceedings of the ACM*

*SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 57–68, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[6] Eugenio Moggi. Computational lambda-calculus and monads. pages 14–23. IEEE Computer Society Press, 1988.

[7] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1989.

[8] Ross Paterson. A new notation for arrows. In *International Conference on Functional Programming*, pages 229–240. ACM Press, September 2001.

[9] R. F. C. Walters. *Categories and Computer Science*. Cambridge University Press, 1992.

# Development of Human Interface Software in our Dental Surgical System based on Mixed Reality

Hiroshi Noborio
Osaka EC University
Kiyotaki 1130-70, Shijo-nawate,
Osaka 575-0063, Japan
nobori@noblab.osakac.ac.jp

Yoshinori Yoshida
Osaka University
Yamadaoka 1-8, Suita,
Osaka, 565-0871, Japan
y-yoshi@dent.osaka-u.ac.jp

Taiji Sohmura
Osaka University
Yamadaoka 1-8
Suita, Osaka, 565-0871, Japan
sohmurat@dent.osaka-u.ac.jp

## ABSTRACT

In this paper, we develop a dental surgical system based on mixed reality, which a dental doctor can scrape a concave tooth with complicated shape by a bar located at the tip of turbine. In this system, we represent a tooth and a dental bar as an octree (a set of voxels) and sets of points, respectively. Based on the octree's hierarchical structure in positioning, we quickly detect an intersection between octree-based tooth and point-based bar. Moreover, according to the intersection set, we scrape a tooth by a bar while making force and moment. Finally, many doctors flexibly pick up visual and tactile parameters according to a lot of their experiences. In addition, our system automatically evaluates a student operation against a professional one by comparing their scraping tooth shapes. For this reason, dental students can learn many kinds of surgical operations on demand via PC and internet.

## Keywords

Mixed reality system, dental surgical simulation, automatic skill evaluation.

## 1. INTRODUCTION

Many kinds of medical or dental surgical education systems have been proposed [Kne03], [DA04], [DD09], [Fra10]. In this paper, we construct a smart but cheaper dental surgical simulation system. Recently, an undergraduate dental student has few chances to experience practices dental surgical operations for patients. Therefore, the system is quite useful for student education.

As very few similar works, we focus on Kim's approach [LSM04], [HLM06a], [HLM06b]: volumetric implicit surface is used for surface modeling and haptic rendering while scraping. The main defective points are as follows: (1) A virtual tooth is roughly approximated, which is not a real patient data captured from a practical CT scanner. (2) Each bar and turbine are roughly expressed by combination of bigger ball and circle pillar. This approximation differs from real kinematic relationship between tooth, bar, turbine which are really used by a dental doctor in a hospital.

On these observations, we propose the other dental surgical simulation system. First of all, all teeth data are captured from real patients, and dental bar, turbine, miller are captured from real dental tools by a dental CT scanner in a hospital. Then, we convert patient tooth and dental bar into an octree (a set of cubes) and a set of points. Then, based on the hierarchical structure of octree representation in positioning, we can quickly check an intersection volume of an octree-based tooth and a point-based dental bar. Using the intersection volume, we simultaneously and precisely calculate force and moment artificially.

Our system is a mixed reality system which consists of visual, tactile, and sound realities. Therefore, we can learn a lot of realistic operation skills on demands. Moreover, our system has two wonderful characteristics as follows: (1) A doctor flexibly and directly selects physical parameters concerning to visual and tactile realities. Needless to say, the doctor has many practical experiments. For this reason, our system's realities are extremely improved. (2) This system automatically evaluates a student's skill by a numerical number between 0 and 100. The numerical point is calculated by comparing two teeth deformed by profession and beginner. Therefore, a student himself can learn several kinds of dental operation skills on-demand without the help of any professional person.

In this paper, we briefly explain our dental simulation system, models of patient teeth and dental tools in section 2. Especially in our system, a dental

doctor uses dental bar and mirror cooperatively. For this reason, our system has two haptics (force feedback devices) PHANToM OMNI. In section 3, we explain how to improve visual reality, and then in section 4, we explain how the doctor selects a lot of parameters for cutting tooth. In section 5, we indicate evaluation functions for two typical operation tasks. Finally in section 6, we conclude this research.

## 2. SYSTEM STRUCTURE

In this section, we firstly describe the system architecture of our dental surgical simulation. Then, we explain dental turbine and mirror in our dental surgical system. In succession, we show models of patient teeth, dental bars, turbine, and mirror.

### Architecture

The reality of our dental simulation comes from the following three aspects [NSK*08a], [NSK*08b]. One is a visual reality designed by the OpenGL. The OpenGL is the most popular API for computer graphics. Another is a sound reality which motor and drill sounds are directly memorized by a high-quality microphone. According to the speed of turbine, we can hear a mixture of rotating motor and drilling tooth, which is controlled by the OpenAL. The other is a tactile reality made from the OpenHaptics. The OpenHaptics, especially, HDAPI (Haptic Device API) is the API for controlling completely a force feedback device PHANToM provided by SensAble Technologies Inc. Our basic software roughly consists of three procedures such as calculating collision check, doing interference volume, and generating force sequence [NK10]. Each force is always felt by the force feedback device PHANToM OMNI based on the OpenHaptics (HDAPI).

Finally, all are combined by Microsoft Foundation Class (MFC) as a human interface (Figures 1 and 2). In our dental simulation system, in order to save production cost, we use a popular PC with normal graphics card, the cheaper haptic (force-feedback) device. This is quite important for constructing an educational system.

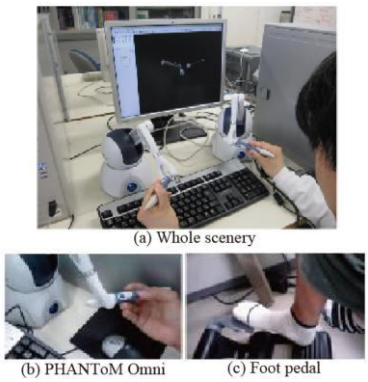



**Figure 1. Panorama of dental simulation system.**

### Turbine and mirror in a dental hospital

In our system, we express all dental data without checking their intersection, for example, dental turbine and mirror by STL. If a dentist treats a molar tooth, he should checks its part by using the dental mirror because he cannot observe the inner part straightforwardly.

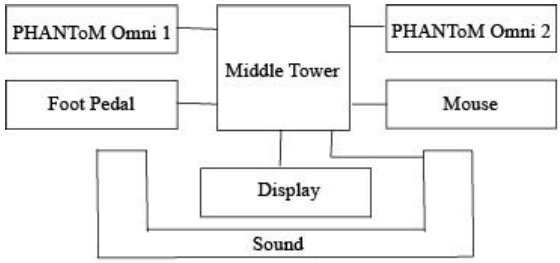



**Figure 2. Architecture of dental simulation system.**

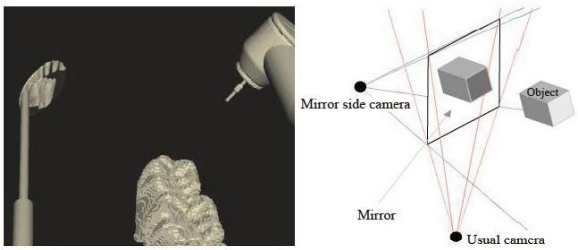



**Figure 3. Cooperative treatment of dental mirror and bar.**

Then, a dentist removes a decayed tooth by a dental bar. This is achieved by two force feedback devises of dental bar and mirror. By using these tools, the dentist can act similar surgical treatments (Fig.3). For mounting this dental mirror, we use the technique of off screen rendering. The technique is as follows: the picture seen from the other side of the dental mirror is the same as the picture that is reflected in the front of the mirror. Therefore, the camera image seen back of the mirror is described in the dental mirror.

### Various models

#### *2.3.1 Model of tooth, turbine, and bar*

In order to capture real shapes of tooth, bar, and turbine in a dental hospital, we use a computed tomography (CT) scan (Figures 4 and 5). It uses X-rays so as to make detailed pictures of structures in and around a human oral. Therefore, we can exactly and individually capture teeth, maxilla, mandible bone, lips, and muscles as several kinds of data structures including STL format. The STL format is one of standard formats for the exchange of surface shape data, especially in rapid prototyping field, which replaces the original surface with a collection of triangulated surface segments. In our system, we precisely use triangular polyhedrons (STL) with the highest resolution 100% from the popular CT scanner

with high precision.

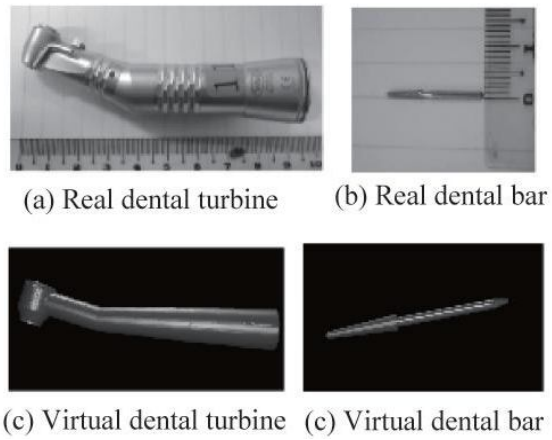



**Figure 4. (a) Real dental turbine, (b) real dental bar, (c) virtual dental turbine, (d) virtual dental bar.**

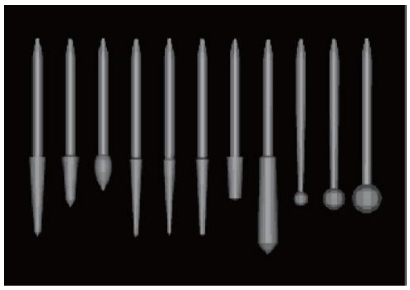


**Figure 5. Many kinds of STL-based bars.**

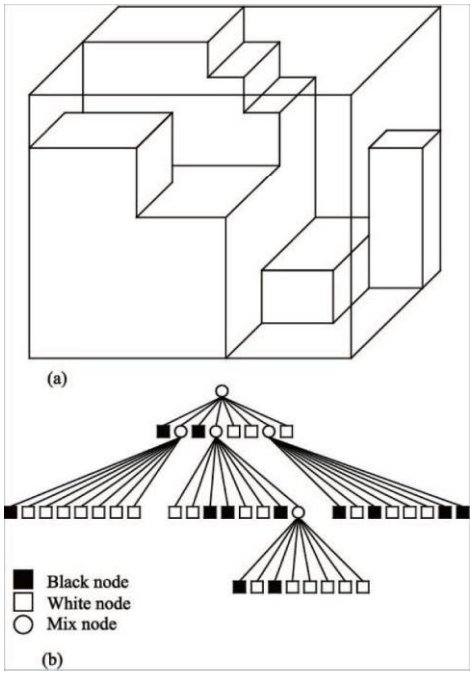



**Figure 6. An octree representing a 3-D environment with objects. It has the hierarchical structure in positioning.**

### *2.3.2 Octree-based tooth model*

In this system, actual STL forms of patient teeth and also dental bars are converted into octree-based teeth and point-based bar, respectively (Fig.6) [JT80]. Based on the hierarchical structure in positioning, the intersection between octree-based tooth and point-based bar can be calculated with high-speed. First of all, we describe a classic method converting the STL format into the octree as follows: If a cube of an octree is inside some convex STL polyhedron, the cube is represented as a black node in the octree. If a cube of an octree is outside all convex STL polyhedrons, the cube is represented as a white node in the octree. Otherwise, the cube is represented as a mix node. Moreover, this procedure is recursively continued for only mix nodes until the octree level corresponds to a given maximum level. The mix node is converted to the black node if the octree level corresponds to the maximum one (Fig. 6).

This classic method is quite time consuming because a 3-D concave STL tooth should be converted into a set of 3-D convex STL parts. To overcome this defective problem, we propose a new method using GPU (Graphics Processing Unit). In general, the STL format can be efficiently converted into octree at high speed by using GPU (Graphics Processing Unit) (Figures 7 and 8). The GPU's Z buffer and parallel schemes quickly convert the STL polyhedral model into a set of 3-D hexahedra as a raster scanner, and then the hexahedron set is easily converted into the octree (Fig.9).

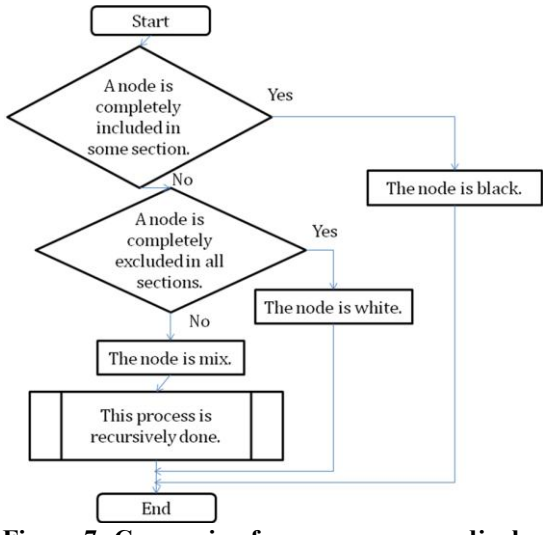



**Figure 7: Conversion from many perpendiculars of an object to an octree.**

We show row data of STL teeth before conversion, and numbers of octree nodes, their memory storage (Bytes) after conversion in Tables 1, 2 and 3, respectively. Here, as a pre-processing in our system, we indicate conversion time from STL-based teeth to

octree-based teeth by GPU (Fig.10). Here the accuracy is 20μm in case of the octree with level 9, the accuracy is 10μm in case of the octree with level 10. As shown in Fig.10, we understand that the computation complexity (processing time) directly depends on not the number of STL patches but the octree level. As a result, we usually use the most precise 100% STL teeth in our research.

As contrasted with this, each dental bar is firstly converted from STL form into an octree, and then is converted from the octree into a set of points which are vertices of octree cubes (Fig.11). Furthermore, we note that those of dental bar are relatively small enough. Therefore, all are completely stored in the main memory in a personal PC.

| Teeth | Patch number | Vertex number |
|---|---|---|
| 6% STL | 8,996 | 26,988 |
| 12% STL | 18,016 | 54,048 |
| 25% STL | 36,036 | 108,108 |
| 50% STL | 72,096 | 216,288 |
| 100% STL | 144,216 | 432,648 |

**Table 1. Numbers of patches and vertices of row STL tooth.**

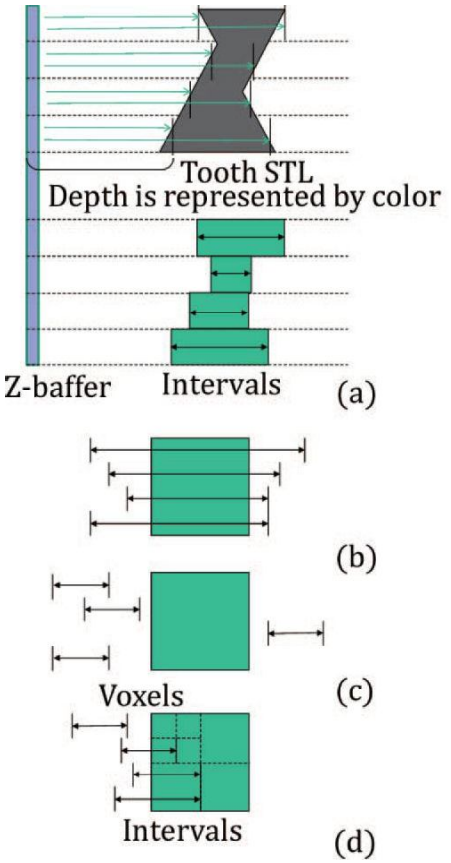



**Figure 8. Conversion from a concave STL polyhedron to an octree via many digital intervals by GPU. (a) Depth calculation and interval collection of STL by GPU. (b) If a cube is to be inside its intersecting intervals, we give a black node for the cube. (c) If a cube is to be outside its intersecting intervals, we give a while node for the cube. (d) Otherwise, we give a mix node for the cube.**

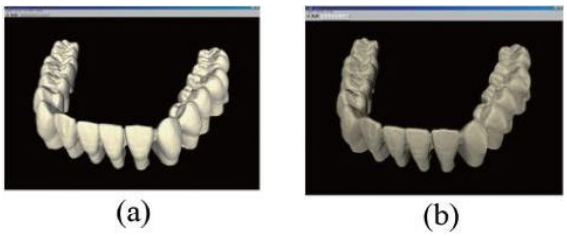



**Figure 9. (a) STL-based teeth before conversion. (b) Octree-based teeth after conversion.**

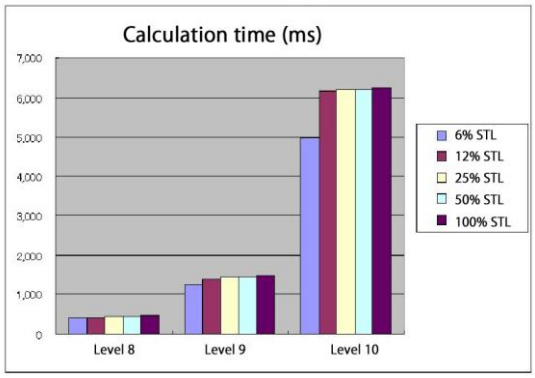



**Figure 10. Conversion time from STL-based teeth to octree-based teeth by GPU.**

### *2.3.3 Multi-layers structure*

Each tooth has the multi-layers structure of enamel, dentin, dental pulp, and dental caries [GH91]. In order to improve the visual reality, a dental doctor easily chooses these colors. In addition, the doctor can select those spring, damper, friction, hardness coefficients so as to improve the tactile reality. The force magnitude and direction depend on an intersection set between point-based bar and octree-based tooth, and substantially depends on the speed before collision (Figures 11 and 12).

| Level | 8 | 9 | 10 |
|---|---|---|---|
| 6% STL | 135,649 | 549,209 | 2206,913 |
| 12% STL | 136,425 | 553,369 | 2214,377 |
| 25% STL | 136,377 | 552,633 | 2212,721 |
| 50% STL | 136,385 | 552,049 | 2214,505 |
| 100% STL | 136,441 | 552,489 | 2213,633 |

**Table 2. Numbers of nodes in the octrees after conversion.**

| Level | 8 | 9 | 10 |
|---|---|---|---|
| 6% STL | 4,595,712 | 19,292,160 | 77,508,608 |
| 12% STL | 4,624,384 | 19,378,176 | 77,545,472 |

| | | | |
|---|---|---|---|
| 25% STL | 4,624,384 | 19,349,504 | 77,492,224 |
| 50% STL | 4,624,384 | 19,333,120 | 77,549,568 |
| 100% STL | 4,624,384 | 19,349,504 | 77,520,896 |

**Table 3. Memory storage of the octrees after conversion (Bytes).**

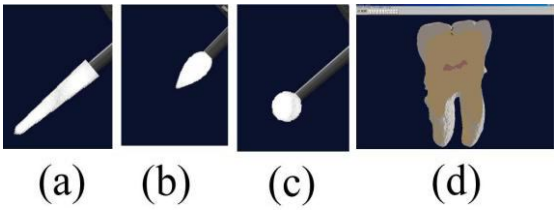



**Figure 11: (a),(b),(c) Many kinds of point-based bars. (d) Tooth model of Multi-layers structure.**

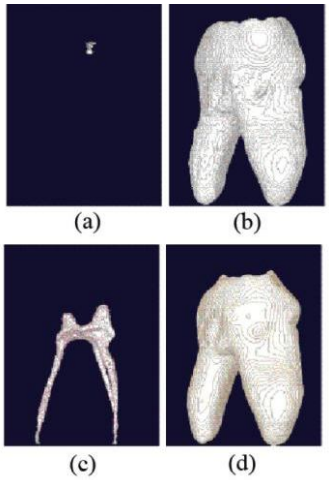



**Figure 12: (a) Octree-based decayed tooth, (b) enamel material, (c) ivory material, (d) dental pulp in a back tooth (molar).**

## 3. IMPROVEMENT OF VISUAL REALITY

In the research, a high-speed collision-check algorithm selects the intersection between octree-based tooth and point-based bar. The octree-based tooth consists of many cubes, and therefore their visibility is bad. To overcome this problem, we firstly used the traditional marching cubes algorithm [LC87]. However, since the visible reality is still not good, we modify the traditional algorithm as follows. In the classic marching cubes algorithm, the normal vector is determined by considering the XYZ direction neighborhood $2^3 = 8$ voxels of an arbitrary voxel. Therefore, in order to improve the reality of visibility, the normal vector is determined by considering the neighborhood $4^3 = 64$ voxels, and also the normal vector is determined by considering $6^3 = 216$ voxels (Fig.13).

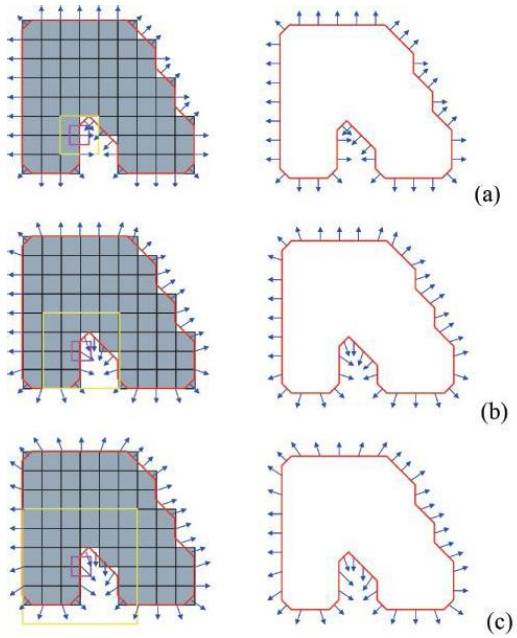



**Figure 13. (a),(b),(c) Normal vector normalization by averaging $2^3 = 8$, $4^3 = 64$, $6^3 = 216$ neighbors in our modified marching cubes method.**

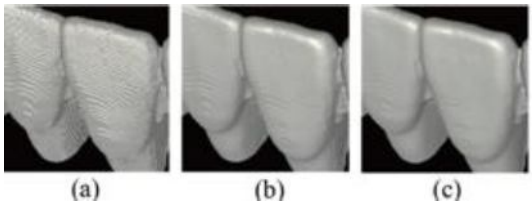



**Figure 14. (a),(b),(c) Three images of octree-based teeth by the modified marching cubes method under averaging $2^3 = 8$, $4^3 = 64$, $6^3 = 216$ neighbors.**

As a result, the reality is quite improved (Fig.14), and also the calculation time is almost less than the video rate. However, the time is sometimes over depending on the operation (conversion of part of $1/32^3$ in whole space) as shown in Fig.15.

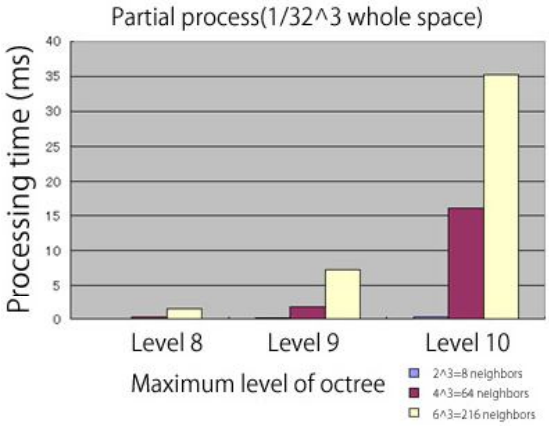



**Figure 15. Calculation time of normal vector at each octree level by the modified marching cubes (Partial conversion of $1/32^3$ whole space).**

## 4. PHYSICAL PARAMENTER SELECTION

In our software, a dentist can flexibly selects many physics parameters concerning to tactile and visual realities. For this purpose, we prepare a wonderful window used by the dentist (Fig.16). First of all, a doctor selects one of "enamel", "dentin", "dental pulp", "dental caries" files in order to choose its spring, damper, frictional coefficients, and hardness (how to drill a tooth easily). They are physics parameters for tactile feeling.

Here, the doctor can select the sense for cutting a tooth by (how much a tooth part is easily drilled) * (how much a dental bar drills easily) * (rotational speed of bar). In particular, the hardness is selected between 0 (not easy to drill) and 1 (easy to drill). We set enamel, dentin, dental pulp and dental caries as 0.1, 0.1, 1.0, 0.5, respectively, as the initial values (Fig.16).

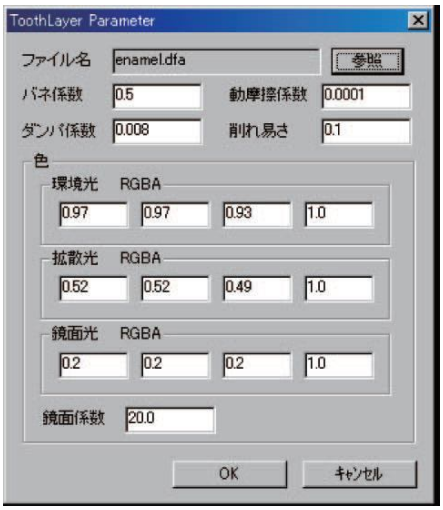



**Figure 16. A window for determining several physical parameters concerning to visual and tactile realities.**

On the other hand, the doctor selects a drill cutting ability between 0 (not easy to drill) and 1 (easy to drill). Finally, rotational speed of dental bar can be controlled between 0: 0rpm - 1: about 40000rpm by stepping or releasing the foot pedal.

Moreover, concerning to physics parameters of visual reality, the dentist himself straightforwardly selects RGBA values according to his real experiences. R is red, G is green, and B is blue of three primary colors. The A means the transparency. In addition, he chooses "ambient: the light that came from all directions at a time", "diffuse: reflect in all directions because of the coming light", "specular: light that formed a bright, white spot to the reflection side".

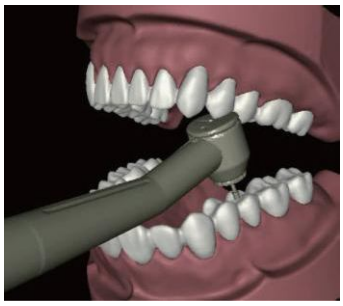


**Figure 17.Cavity preparation.**

**Figure 18. Automatic evaluation system.**

## 5. STUDENT OPERATION EVALUATION

This system automatically evaluates student's skills in dental surgical operation (Figures 17 and 18). Concerning to this, we explain how to evaluate student operation in two typical dental tasks as follows (Full marks are 100 point).

(a) Caries removal task (Evaluation whether the caries was smoothly removed or not) (Figures 19 and 20)

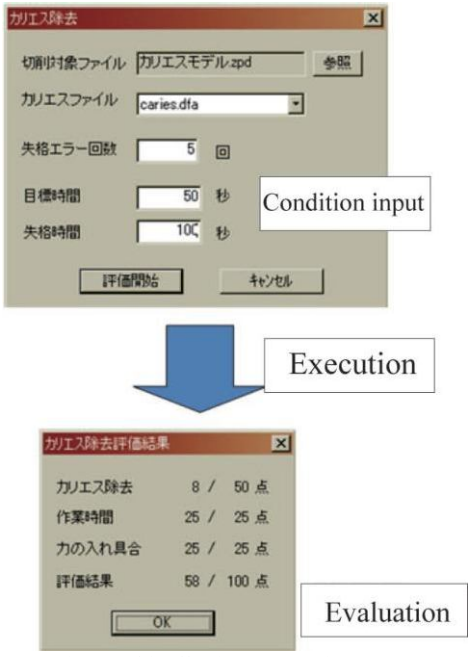



**Figure 19. Caries removing task.**

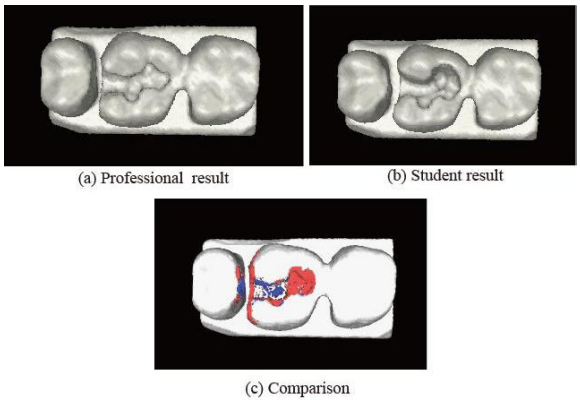



**Figure 20. (a) Professional result, (b) Student result, and (c) over and under cutting regions are colored by red and blue, respectively, for caries removing task.**

Measurement contents are as follows: Numbers of two kinds of force errors, initial and final caries volumes, start, final, target, and failure time. This detail is described in Table 4.

| No. | Item name | Detail |
|---|---|---|
| 1 | Score of removing caries | = (Final caries volume) / (Initial caries volume)*50. Full marks are 50 points. It is expressed by the decimal point. |
| 2 | Score of executing time | = (failure time - (finish time-start time)) / (failure time- target time) * 25. Full marks are 25 points. Less than 0 is expressed as 0 point, and more than 25 is represented as 25 points. All are represented by the decimal point. |
| 3 | Score of controlling forces | = (The number of forces more than a given threshold – the number of relatively larger forces) / (The number of forces more than a given threshold) * 25. Full marks are 25 points. Less than 0 is expressed as 0 point, and more than 25 is represented as 25 points. All are represented by the decimal point. |
| 4 | Total score | = Score of removing caries (50 points) + Score of executing time (25 points) + Score of controlling forces (25 points). Full marks are 100 points. |

**Table 4. Evaluating terms for caries removing task.**

(b) Cavity preparation task (Evaluation whether the ideal cavity was built or not) (Fig.21)

Measurement contents are as follows: Under and over cutting volumes, number of force errors, start, final, and target time. This detail is described in Table 5.

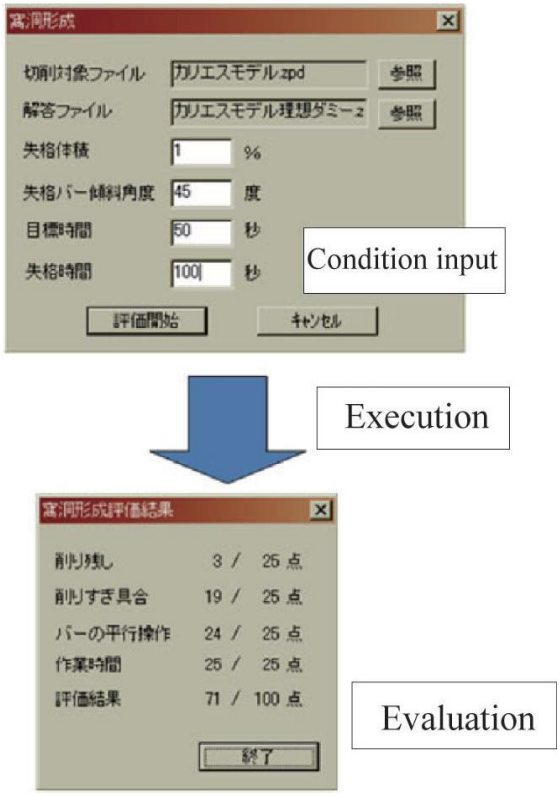



**Figure 21. Parameter setting and operation evaluating for the cavity deforming task.**

## 6. SUMMARY

In this paper, we explained our dental surgical simulation system for educational purpose. Especially, we illustrate how to obtain visual and tactile realities, and also show automatic evaluation of student surgical skills. We ascertained that our system is superior to previous similar ones concerning to these activities. This is achieved as follows: (1) our system was compared with other commercial or non-commercial surgical systems in several workshop demonstrations. (2) our system is carefully testing and investigating by over 100 professors and students in the faculty of dentistry.

For the education purpose, we frequently heard that ten cheaper systems (For example, one million Japanese yen) are better than one expensive system (For example, 10 million Japanese yen). Because the system consists of commercial-based PC and graphics acceleration board, and the cheapest haptic device, the cost is less than fifty hundred thousand yen.

Furthermore, we use open source software in visual, tactile, and sound realities as OpenGL, OpenHaptics, and OpenAL, respectively. Therefore, our dental surgical simulation system has high extendibility. As a result, our system has a wonderful potential for education purpose.

| No. | Item name | Detail |
|---|---|---|
| 1 | Score of under cutting the ideal cavity | = Under cutting volume * demerit mark coefficient * 100 / cutting volume proposed by profession. Under cutting volume is digitally calculated as (final volume **AND** (**NOT** ideal cavity volume by profession)). Full marks are 25 points. |
| 2 | Score of over cutting the ideal cavity | = Over cutting volume * demerit mark coefficient * 100 / cutting volume proposed by profession. Over cutting volume is digitally calculated as ((**NOT** final volume) **AND** ideal cavity volume). Full marks are 25 points. |
| 3 | Score of parallel operation of bar | If a dental doctor operates a dental bar in parallel, its operation force constantly decreases. For this reason, we indirectly evaluate the number of parallel operations by the number of relatively larger forces. As the larger the number of relatively larger forces* force error coefficient, the larger the score is. A threshold of force error is given in advance such as 3.0 N. Full marks are 25 points. All are represented by the decimal point. |
| 4 | Score of executing time | = (executing time - target time) * executing coefficient / 60. Executing time = final time-start time. 60 means the translation parameter from seconds to minutes. Full marks are 25 points. All are represented by the decimal point. |
| 5 | Total score | = Score of under cutting the ideal cavity (25 points) + Score of over cutting the ideal cavity (25 points) + Score of parallel operation of bar (25 points) + Score of executing time (25 points). Full marks are 100 points. |

**Table 5. Evaluating terms for cavity deforming task.**

## 7. ACKNOWLEDGMENTS

This is supported in part by 2006 and 2010 Grants-in-aid for Scientific Research (No.18360128 and No.22360109) and also is supported in part by the 2007 Modern Good Practice, from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

## 8. REFERENCES

[Kne03] KNEEBONE R.: Simulation in surgical training: educational issues and practical implications. Journal of Medical Education (2003), pp.267–277, vol.37, no.3.

[DA04] DELINGETTE H., AYACHE N.: Soft tissue modeling for surgery simulation. Computational Models for the Human Body. N. Ayache, ed., Handbook of Numerical Analysis, P. Ciarlet and N. Ayache, eds., Elsevier (2004).

[DD09] DIBART S., DIETRICH T.: Practical periodontal diagnosis and treatment planning. Wiley-Blackwell (2009).

[Fra10] FRAUNHOFER J. A.: Research writing in dentistry. Wiley-Blackwell (2010).

[LSM04] L.KIM, S.G.SUKHATME, M.DESBRUN: A haptic rendering technique based on hybrid surface representation. IEEE Computer Graphics and Applications (2004), pp.66–75, vol.24, no.2.

[HLM06a] YAU H.T., TSOU L.S., TSAI M.J.: Haptic interaction and volume modeling techniques for realistic dental simulation. The Visual Computers (2006), pp.90– 98, vol.22, no.2.

[HLM06b] YAU H.T., TSOU L.S., TSAI M.J.: Octree-based virtual dental training system with a haptic device. Computer-Aided Design and Applications (2006), pp.415–424, vol.3, nos.1–4.

[NSK*08a] NOBORIO H., SASAKI D., KAWAMOTO Y., TATSUMI T., SOHMURA T.: Mixed reality software for dental simulation system. Proc. of the 7th IEEE Int. Workshop on Haptic Audio Visual Environments and Games (2008), pp.19–24.

[NSK*08b] NOBORIO H., SASAKI D., KAWAMOTO Y., SOHMURA T., "Construction of Dental Simulation System with Mixed Visual, Tactile, and Sound Realities," Proc. of the 18th Int. Conf. on Artificial Reality and Telexistence, Tokyo, Japan, December 1-3, (2008), pp.93-100.

[NK10] NOBORIO H., KAWAMOTO Y.: Digital collision checking and scraping tooth by dental bar. Proc. of the 2010 IEEE RAS/ EMBS Int. Conf. on Biomedical Robotics and Biomechatronics, (2010), to appear.

[JT80] JACKINS C.L. and TANIMOTO S.L., "Octtrees and their use in representing three-dimensional objects," Computer Vision, Graphics, Image Processing, (1980), pp.249-270, vol.14, no.3.

[GH91] GALYEAN T.A. and HUGHES J.F., "Sculpting: an interactive volumetric modeling technique," Computer Graphics, July (1991), pp.267-274, vol.25 no.4.

[LC87] LORENSEN W. E. and CLINE H. E.: Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics, July (1987), pp.163 – 169, vol. 21, no. 4.

# Metaphorical Visualizations of Graph Structures

Ľuboš Ukrop Martin Jakubéci Peter Kapec
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3, 842 16, Bratislava, Slovakia
{lukrop, matko.jaka}@gmail.com, kapec@fiit.stuba.sk

## ABSTRACT
Data visualization of large abstract data sets and complicated relations is a complex research area with different problems and constraints. Often simple shapes and structures are not very eye-pleasing. Visualization metaphors, which create a mapping between a well-known problem domain and a new complicated problem domain, can produce interesting visualizations. In this paper we propose two metaphorical visualizations of graphs and multidimensional data: we propose a metaphor of soap bubble clusters to visualize graphs and a nebulae (sky) metaphor, which uses nebulae and stars to visualize graphs and multidimensional data.

## Keywords
soap bubbles, nebulae, visualization metaphor, graph visualization, hypergraph

## 1. INTRODUCTION
Complex data structures in their raw or pure textual form are unnatural for human perception system and can be very difficult to understand. To enhance comprehensibility, such data are often presented in visual form and represented by abstract geometrical shapes. This idea is further extended by metaphoric visualizations that use the analogy with real world objects. Visualization is often divided into scientific and information visualization, however there is a potential overlap, especially when considering metaphorical visualizations. In this paper we present two metaphorical visualizations of graph structures and multidimensional data, which were inspired by real world phenomena: soap bubbles and nebulae with stars.

In Section 2 we outline basic information from the data visualization field related to our work. Section 3 describes proposed technique of graph visualization using soap bubbles metaphor and Section 4 introduces a metaphoric visualization that uses visual syntax of stars and nebulae to create unconventional presentations of graphs and multidimensional data. The proposed visualization metaphors are illustrated by visualizations of real data sets. Section 5 discusses related works and is followed by conclusions.



## 2. DATA VISUALIZATION
Data is the main object of interest in the visualization. The goal of information visualization is to display abstract entities and relations to provide better insight and easier understanding of information. Important data types that are used in many areas are graphs and multidimensional data. Both require specific approaches for visualization.

### Graph visualization
Data, which are decomposable to elements and relations between them, can naturally be represented by graphs. Human understanding of graphs is enhanced by its visualizations. Graph visualization deals with creating, presenting and navigating through graphical representations of graphs. Numerous graph visualization algorithms have been developed (see [Her00] for survey in this area). Most of them are specialized according to the type of input graph – there are algorithms for visualization of trees, oriented graphs, hypergraphs etc. Problems and issues of graph visualization are mostly related to the size of input graph. While trying to visualize large graphs, we have to deal not only with the limits of displaying platform, but also with the limits of human perception system.

First step towards successful visualization is layouting of graph elements. Graph nodes and edges are usually placed in two dimensions, though the usage of 3D or even non-Euclidean geometry is becoming also common. To produce comprehensible output, the layout process has to follow certain aesthetic criteria [Pur00]. Popular methods to solve this task are force directed layout algorithms that employ a physical model built according to the graph structure. Starting from random initial node

placement, simulated forces acting between graph's nodes transforms the layout toward a state with minimal energy. This method was originally proposed by Eades [Ead84], but there are also many other popular variations, e.g. by Fruchterman and Reingold [Fru91]. After the layout is determined, nodes and edges are replaced with their graphical representations usually with the respect to the conventions accepted in this area.

## Visualization of multidimensional data

Visualizing multidimensional data (data with more than three dimensions) is complicated, because the human eye is only able to recognize three dimensions. The best examples of multidimensional data are database tables that may contain millions of rows (records) and hundreds of columns (dimensions).

Many different methods to visualize multidimensional data in two or three dimensions have been developed, overview can be found in [Spe99]. Most approaches use some type of mapping attributes to visual and/or structural properties e.g. size, length, color, angle, shape etc. [Sii07]. Very popular are various projection methods that use different kinds of projections from higher dimensions into lower, e.g. Grand tour [Weg92] or SOM [Lat07].

Relatively new approaches use different high-dimensional clustering methods [Ber02]. An interesting approach, related to graphs, is based on clustering using a hypergraph model. During the first step, a weighted hypergraph is constructed to represent the relations among different items, and during the second step, a hypergraph partitioning algorithm is used to find *k* partitions such that the items in each partition are highly related [Han97].

## Visualization metaphors

The world of computers is full of associations and descriptions that are based on appearance similarity, which help us to understand the nature of the problems. Metaphor is a tool, which enables us to do this. Metaphors help us to understand one problem area using another problem area. In general we can say that a metaphor is a projection between the source problem domain and the destination problem domain and we want to understand the destination area by comparison to the source area [Ave08].

Visualization metaphors use this metaphorical projection to visualize data. There are different examples of visualization metaphors: a solar system metaphor that uses stars and planets [Gra04], a city metaphor with buildings [Chi05] or a desktop metaphor [Lar09]. For graph visualizations we can mention the very popular molecule metaphor [Ave08]. Others can be also mentioned, e.g. dashboard metaphor, address metaphor etc.

# 3. SOAP BUBBLES METAPHOR

Soap bubbles are spherical structures made of thin soap fluid film that encloses certain volume of air. Their visual attractiveness and clustering dispositions led us to the idea to utilize them as a visual syntax used for visualization of simple graphs. In this visualization metaphor the graphs are presented as soap bubble clusters with their nodes represented by individual bubbles and edges displayed as cross connections of bubbles that represent the incident nodes. Since the structure of the cluster could be quite dense, connection between not adjacent bubbles will likely occur. Therefore we decided to enhance this visual representation with conventional graphical links used to indicate edges. Textual data attached to graph nodes can be displayed inside the bubbles.

To realize proposed visualization technique three main problems have to be solved: how to layout graph elements, how to represent soap bubble cluster geometry and how to simulate its optical properties to achieve realistic appearance.

## Graph layout

To ensure that layout process will produce 3D structures similar to soap bubble clusters, couple of requirements have to be met. For a pair of adjacent nodes (i.e. bubbles), distance of their centers has to be approximately equal to the radius of the larger bubble. On the other hand, a pair of not adjacent bubbles has to be in maximal correlative distance, while preserving connections with adjacent nodes.

For this purpose, we utilized a custom iterative force directed layout algorithm. Each iteration starts with force accumulation. Force acting is defined between each pair of nodes. Let *i* and *j* be two different nodes, *d* distance between their centers and *dir* unit direction vector from *i* to *j*. Then the force acting between them is calculated as follows:

```
force = 0
IF i and j are adjacent THEN
    delta = d – max (i.radius, j.radius)
    force = dir * springStiff * delta
    IF delta > 0.0 THEN
        force = force * springCoef
ELSE:
    delta = d – (i.radius + j.radius)
    IF delta <= repelThreshold THEN
        force = – dir * repelCoef * exp(– delta)
i.force = i.force + force
j.force = j.force – force
```

Simplified interpretation of this is that adjacent nodes are connected by springs with natural length equal to node's larger radius and repulsive forces act between not adjacent nodes. Default values of parameters of the simulated spring (*springStiff*, *springCoef)*, and repulsing forces (*repelThreshold, repelCoef)* were determined experimentally based on observed behavior. With the mass of nodes assigned proportionally to their radius, accumulated forces are

applied[1], resulting in transition to next spatial configuration. To bring in variability of bubble sizes, which is typical for real clusters, radius is defined proportionally to the node degree using the arcus tangent function.

## Cluster geometry

For the purpose of layout it was sufficient to characterize each bubble only by its position and radius. However, the rendering process requires more detailed description of the bubble surface. Isolated bubbles tend to have regular spherical shape. With bubbles included inside a cluster, situation is more complicated. Their surface is deformed according to cluster structure. In our solution, the structure of bubble cluster is a direct consequence of graph layout. Since the graph layout may change very frequently, we needed a solution that is able to adapt geometrical representation dynamically in shortest possible time. The approach presented by Sunkel et al. [Sun04] is very suitable for dynamic changes.

Bubbles are initially represented by spherical polygonal meshes. In each rendered frame their shape is modified in two steps:

1. For each bubble, bubbles with which it collides are identified. Based on that, list of intersection planes is built. By intersection plane we mean a plane that separates a part of the bubble surface, which is exceeding into another bubble.
2. Vertex shader in GPU accepts the list of intersection planes that belongs to currently rendered bubble. Before each vertex is transformed, it is tested against each intersection plane. In case it is an exceeding vertex, it is projected onto the intersection plane along the normal. This process is illustrated in Fig.1 - vertex P and all other exceeding vertices are shifted to form the junctions.

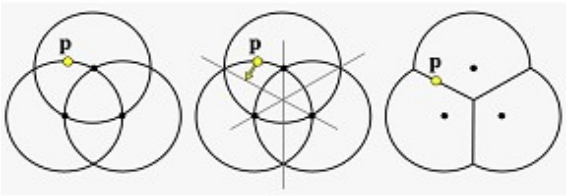



**Figure 1. Bubble collisions resolving.**

Soap bubble clusters produced with this method will not embrace all the geometric properties of a real cluster. However, such realism was not even our intention.

## Visual properties

When the light hits the surface of a soap bubble, several optical effects are observable. Characteristic rainbow-like color toning is caused by interfering light waves. The most distinguished interference occurs between the wave reflected from outer soap film boundary and wave once reflected from inner boundary leaving the film with the same incidence angle. Resulting light intensity $I_r$ is given by following equation:

$$I_r = 4\mathrm{I}_i R(\theta)\sin^2\left(\frac{2\pi}{\lambda} w\eta\cos(\theta_t)\right) \qquad (1)$$

Where $I_i$ is incoming light intensity, $\theta$ is incidence angle, $\theta_t$ is transmitted angle, $R(\theta)$ is reflectance, $\lambda$ is wave length, w is film thickness and $\eta$ stands for soap water index of refraction [Gla00]. Because of Snell's law, the refraction of light also occurs. But since the film thickness is extremely small, it is significant only at the bubble boundaries [Küc02]. Fresnel effect causes, that with declining light incidence angle transparency of the surface raises and reflectivity becomes less obvious.

Photo-realistic rendering could be achieved by precise simulation of all the mentioned optical effects and properties with the use of ray-tracing algorithm. However, since our visualization technique was intended to function on conventional hardware in real-time, we utilized less computationally expensive solutions. Our approach is based mainly on the works of Glassner [Gla00] and Iwasaki et al. [Iwa04]. Using Equation 1 with constant light color (in spectral representation), and similarly to Iwasaki we precompute the interference effect and save it into a 2D texture (see Fig.2a) with vertical coordinate interpreted as the film thickness and horizontal as cosine of incidence angle. Alpha channel contains Fresnel reflectivity used by alpha blending transparency implementation. Film thickness is preserved in a grayscale texture that is mapped directly on the bubble surface (see Fig.2b).

Final soap film color calculation takes place in a fragment shader and involves texels from interference texture (which are computed based on film thickness texture and cosine of incident light angle), texels from environment cube map and diffuse material color. Reflections of dynamic objects (i.e. bubbles) and refraction are omitted.

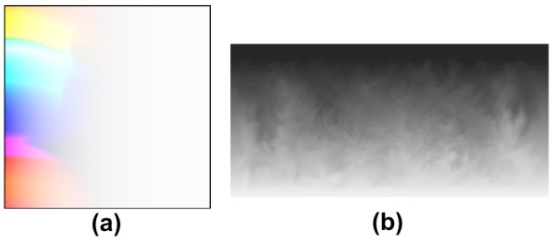



**Figure 2. Interference (a) and thickness (b) texture.**

Using the proposed visualization technique we were able to interactively visualize smaller graphs (approx. up to 100 nodes). Fig.3 shows visualization of

[1] We utilized Bullet physical engine for this purpose - http://bulletphysics.org.

a graph that exposes relations between scientific disciplines.

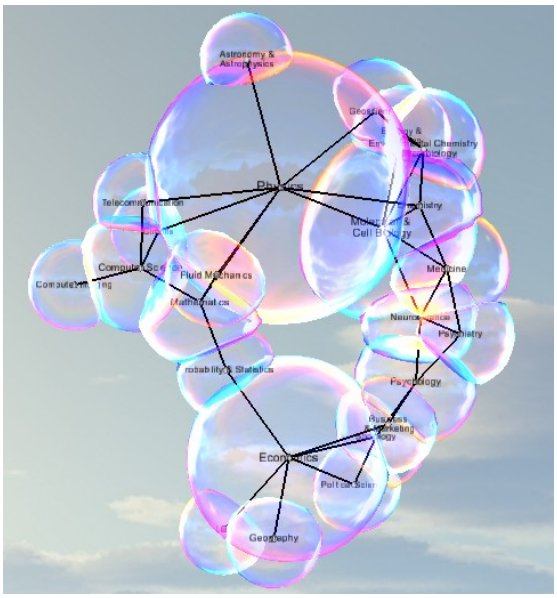



**Figure 3. Sample graph visualized by soap bubbles metaphor.**

The soap bubbles metaphor increased visual attractiveness of visualization, especially comparing to traditional node-link drawings, though the readability (especially when presented by static image) was reduced. To enable user to directly change the shape of a cluster, mechanism of space constraints was introduced. Space constraints act as enclosed barriers that the bubbles do not manage to cross. Effect of visualization constrained by flatten box is shown in Fig.4.

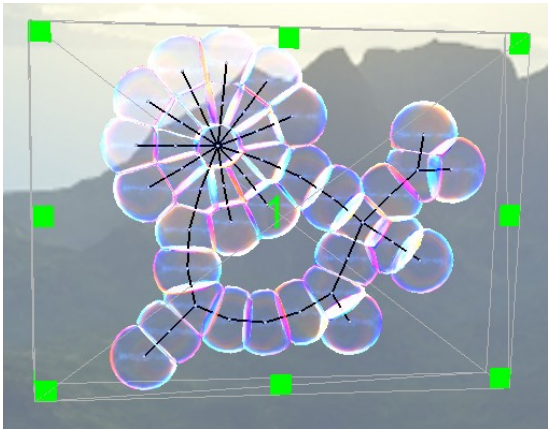


**Figure 4. Visualization with space constraint.**

Fig. 11 demonstrates visualization of concrete data. It is a part of graph obtained by extraction of software artifacts from source code of real application[2]. It contains all the extracted functions (left part of the cluster) with one of them represented in detail, with its parameters and associated comments (right part of the cluster).

[2] LuaDist - http://www.luadist.org

## 4. NEBULA METAPHOR

Nebulae (or sky) metaphor is used to visualize data using objects from night sky (from the universe). Typical objects from the universe are stars, nebulae, galaxies etc. The main advantage of this metaphor is that these objects are well known by all people, including small children.

In this approach, stars are used to represent entities (nodes of a hypergraph) and nebulae are used to represent relations (hyperedges of a hypergraph). The stars can be drawn as 3D shapes, for example spheres or textured rectangles, called billboards. Nebulae can be drawn using volumetric rendering or using particle system. Volumetric rendering gives very realistic results [Nad00], but its computational complexity makes it unusable in the case of data visualization. The reason is that real-time drawing and modification of many nebulae is needed. Particle systems are a better solution, but still hard to control in real-time and to draw hundreds of nebulae. A modification of particle systems is used, with pre-rendered cloud particles on a texture, which is mapped on rectangle, so the nebulae are drawn using billboards as well.

The next problem is how to place the billboards in space. A hyperedge visits many nodes and the nebula has to be placed between these nodes. In our approach, a center of the hyperedge is computed and the middle of a billboard is placed on the middle of a line segment, which connects the center of the hyperedge with a node. Billboards are generated for every node a hyperedge visits. Fig. 5 shows a hyperedge with three nodes (circles) and its center (triangle) and how the corresponding three billboards are placed (red, green and pink rectangles).

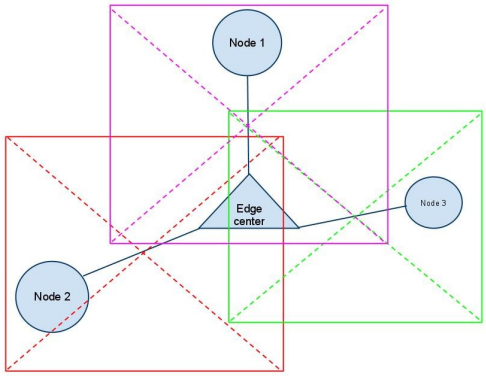



**Figure 5. Nebulae billboards placement.**

To distinguish between different hyperedges, the color of the nebulae has to be modified. This is achieved by setting the color of the rectangle to a color with alpha channel and then the color of the rectangle is added to the cloud texture using alpha blending. The colors of nebulae seen in space are quite specific, so a set of colors shown in Fig. 6. was chosen. Also other colors-sets can be used, however their selection, probably related to data, can produce

familiar e.g. clouds/smoke-like visualizations or completely unfamiliar visualizations.

**Figure 6. Nebulae colors [Fad05].**

Now that we prepared a metaphorical drawing method, we have to prepare a hypergraph layout. The standard way to do this is to convert hypergraph into a bipartite graph and then use a graph layout algorithm. In our approach, the hypergraph is layouted and drawn in its pure form. We use a modified Fruchterman-Reingold algorithm [Fru91], where the nodes of a hypergraph are attracted to the center of the hyperedge. After the attractive and repulsive forces are computed and applied, the center of every hyperedge is recomputed as a center of mass, by computing the average position of all node positions, which are connected by the hyperedge.

Fig. 7 shows a hyperedge, which represents calling of a method in software artifacts data. This hyperedge connects eight nodes, which represent different parameters. The hyperedge is visualized using a snebula with eight billboards with cloud textures and the nodes are visualized using billboards with a star texture.


**Figure 7. Hyperedge visualized using nebulae metaphor.**

Fig. 8 shows a hypergraph with several hyperedges, which were obtained from a larger hypergraph (Fig. 12) by filtering. Hyperedges are visualized using nebulae and stars, but with lines and captions disabled.

To demonstrate the metaphorical visualization a real data set was used. Fig. 12. presents a visualization of software artifacts from a real software system, which was implemented using Lua scripting language. The data includes different types of methods, classes, documentation relations etc. It consists of 1233 nodes and 459 hyperedges.

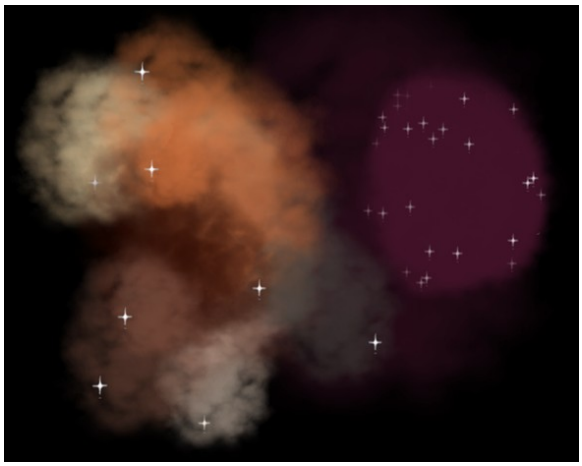


**Figure 8. Hypergraph visualized using nebulae metaphor.**

## Hypergraph based multidimensional clustering

To visualize multidimensional data a simple clustering method inspired by the work [Han97] was implemented. This method utilizes the hypergraph representation and layout. The multidimensional data is transformed into hypergraph structure and then the clusters are automatically created by the layout algorithm. Transforming multidimensional data (rows and columns) can be done in different ways; we used classification of numerical values into ranges or intervals. The clusterisation process is done in these steps:

1. Create an empty hypergraph.
2. Create a node for every row.
3. Create intervals for every column.
4. Create a relation between every value of a row and the corresponding interval.
5. Create a hyperedge for every interval of every column, which is connected by a relation from step 4 to at least two rows. This hyperedge consists of nodes that were created in step 2 and are connected by a relation from step 4.
6. Apply the layout algorithm to the hypergraph.

Fig. 9 shows, how it is done on a sample data set, with six rows and two columns. One interval is created for every column (red interval for first column, green interval for second column); the other values are too different, so no other intervals are needed. Graphical representation shows the generated hypergraph, blue circles are rows, green and red circle are hyperedges, which represent intervals.

Steps 3, 4 and 5 of the process are a little bit complicated. Intervals in step 3 can be created by

different methods, we use a simple approach, where values of the columns are iterated and the minimum and maximum values are found. Then the range between these two values is divided into 10 equal intervals.

| Name | C1 | C2 |
|---|---|---|
| R1 | 24 | 2 |
| R2 | 45 | 3 |
| R3 | 145 | 2 |
| R4 | 2 | 1 |
| R5 | 3 | 1567 |
| R6 | 5397 | 538 |

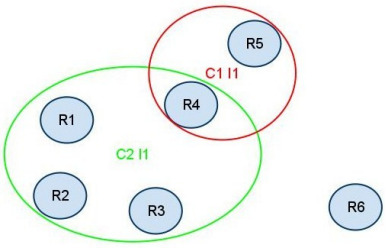



**Figure 9. Clustering sample.**

After that, we have to find out, to which interval every value of the columns fits. This is done in step 4. Then we just create a model of this situation by representing the relation between a value and its corresponding interval, by creating hyperedges. So rows with similar values of a column (values in the same interval) are connected together by a hyperedge and attracted by applying the layout algorithm. Fig. 10 shows a clustered dataset with different information about proteins. It consists of 1484 records and 8 columns (dimensions) and in the clustering process 1484 nodes and 80 hyperedges (intervals) were created[3].

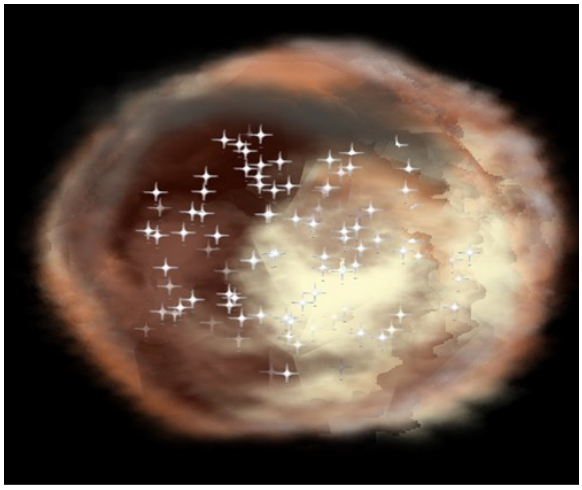


**Figure 10. Clustered protein data visualized using nebulae metaphor.**

[3] Data-set from http://archive.ics.uci.edu/ml/datasets.html Machine learning repository of University of Carolina

## 5. RELATED WORK

There were numerous attempts of computer simulation and rendering of soap bubble clusters. Glassner [Gla00] uses a sequence of CSG operations to create a cluster model considering geometric properties of real soap bubble clusters. With precise optical calculations implemented in a fragment shader he was able to achieve high level of realism, but his cluster consists of only three bubbles (analytical solutions exist merely up to this number) and relatively high computational complexity makes his solution inappropriate for real-time rendering. Ďurikovič targeted mainly dynamics of soap bubble clusters [Ďur05]. He represents each bubble as a system of particles connected by springs, taking all significant forces into account, thus simulating bubble creation, coalescence and bubble-plane collisions. Ďurikovič did not describe optical properties. Iwasaki et al. [Iwa04] combined mentioned works to simulate and render a small number of bubbles in real-time on conventional hardware. They reduced computational complexity mainly using a precomputed interference effect. Kück et al. [Küc02] developed rendering and simulation technique for large liquid foam structures. The foam is represented by set of spherical polygonal meshes connected by virtual springs. Junctions between colliding foam bubbles are computed directly inside the shader for ray tracing based renderer. Sunkel et al. [Sun04] came with real-time simplified simulation of large liquid foams: they are creating approximated planar bubble junctions in vertex shader by shifting overlapping vertices to the plane of intersection. Mentioned works were done only with an intention to handle this natural phenomenon by the means of computer science, without presenting concrete practical applications of their results. We applied simulation of soap bubbles in the area of data visualization to create a novel graph visualization technique, while using existing approaches from both fields (mainly [Gla00], [Iw04] and [Sun04]).

Few works use visualization metaphors similar to the nebula metaphor. A sky metaphor was used to visualize self-organizing maps [Lat07]. It displays data records as stars and the clustering process creates star clusters. A visual-analytic tool called IN-SPIRE and its predecessor SPIRE use a galactic metaphor [Won04]. The visualization is using stars and star clusters to help in analyzing of large data. Info-Vis visual explorer is used to interactively explore large collections of documents, which are displayed as stars and collections in the hierarchy are visualized as bounding polygons [Gra07]. All of these works are using just stars to show data records and mostly use star clusters to demonstrate similarity. Our approach is using a similar method when showing clustered multidimensional data, but is also able to show complicated relations displayed as nebulae. It is the only approach, which utilizes 3D visualization as well.

## 6. CONCLUSIONS

In this paper we presented two visualization metaphors applicable on structured data. Soap bubbles metaphor was utilized to create an experimental technique of graph visualization using 3D soap bubble clusters. Based on existing methods and approaches for computer simulation of soap bubbles and custom force directed layout, it enabled us to interactively visualize smaller graphs in real-time. Second presented technique uses nebulae metaphor for visualization of hypergraphs in 3D. With visual syntax of stars and colored nebulae, it is able to visualize also multidimensional data, which can be transformed to hypergraph representations. Both proposed metaphors offer an interesting and unconventional data presentation. Future work will be dedicated to verifying practical usability by the means of user testing, quantitative evaluation and comparison with standard visualization techniques. Both presented metaphoric visualization are suitable for further experiments, e.g. adding smoke into bubbles or applying solar winds to nebulae.

## 7. ACKNOWLEDGMENTS

This work was supported by grant KEGA 244-022STU-4/2010: Support for Parallel and Distributed Computing Education and grant VEGA 1/0848/08: Connectionist Computational Models for Computer Grid Environment.

## 8. REFERENCES

[Ave08] Averbukh V.L. et al. Searching and analysis of interface and visualization metaphors. Human-Computer Interaction, New Developments, Vienna, pp. 49-84, 2008.

[Ber02] Berkhin, P. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, 2002.

[Chi05] Chiu, P. et al. MediaMetro: browsing multimedia document collections with a 3D city metaphor. In: Proc. of the 13th ACM international conference on Multimedia, pp. 213-214, 2005.

[Ďur05] Ďurikovič, R. Animation of soap bubble dynamics, cluster formation and collision. Journal of the Applied Mathematics, Statistics and Informatics, vol. 1, no. 2, pp. 33-48, 2005.

[Ead84] Eades, P. A heuristic for graph drawing. Congressus Numerantium, vol. 42, no. 1, pp. 149-160, 1984.

[Fad05] Fadai, K. Painting a Nebula. Artistic tutorial, 2005.

[Fru91] Fruchterman, T., Reingold, E. Graph drawing by force-directed placement. Software-Practice & Experience, vol. 21, no. 11, pp. 1129-1164, 1991.

[Gla00] Glassner, A. Soap Bubbles: Part2. IEEE Computer Graphics and Applications, vol. 20, no. 6, pp. 99-109, 2000.

[Gra04] Graham, Y.H. et al. A solar system metaphor for 3D visualization of object oriented software metrics, In Proc. of the Australasian Symposium on Information Visualization, Australian Computer Society, Inc., pp. 53–59, 2004.

[Gra07] Granitzer, M. et al. InfoSky. InfoVis Wiki.

[Han97] Han, S. et al. Clustering in a high-dimensional space using hypergraph models. Technical report, Department of computer science, University of Minnesota, 1997.

[Her00] Herman, I. et al. Graph visualization and navigation in information visualization: A Survey. IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 1, pp. 24-43, 2000.

[Iwa04] Iwasaki, K. et al. Real-time rendering of soap bubbles taking into account light interference. In: Proc. of the Computer Graphics International (CGI'04), pp. 344-348, 2004.

[Küc02] Kück, H. et al. Simulation and rendering of liquid foams. In: Proceedings of Graphics Interface, pp. 81-88, 2002.

[Lar09] Lardinois, F.: Bumptop launches: make your physical desktop virtual. Blog, ReadWriteWeb, 2009.

[Lat07] Latif, K. and Mayer, R. Sky-metaphor visualisation for self-organising maps. Journal of Universal Computer Science, Proc. of 7th International Conference on Knowledge Management, pp. 400-407, 2007.

[Nad00] Nadeau, D. R. et al. Visualizing Stars and Emission Nebulas, In: Proc. of Eurographics, Eurographics Association, 2000.

[Pur00] Purchase, H. C. A study of graph drawing aesthetics and algorithms. Interacting with Computers, vol. 13, no. 2, pp. 147-162, 2000.

[Sii07] Siirtola, H. Interactive visualization of multidimensional data. PhD dissertation, University of Tampere, 2007.

[Spe99] Spears, W.M.: An overview of multidimensional visualization techniques, in evolutionary computation, Morgan Kaufmann, pp. 104-105, 1999.

[Sun04] Sunkel, M. et al. Rendering and simulation of liquid foams. In: Proc. of the Vision, Modeling and Visualization, pp. 285-294, 2004.

[Weg92] Wegman, E.J. The Grand Tour in k-Dimensions. Computing Science and Statistics, In: Proc. of the 22nd Symposium on the Interface, Springer-Verlag, pp. 127-136, 1992.

[Won04] Wong, P.C. et al. IN-SPIRE InfoVis 2004 Contest Entry. In: Proc. of the IEEE Symposium on Information Visualization, pp. 216–217, 2004.

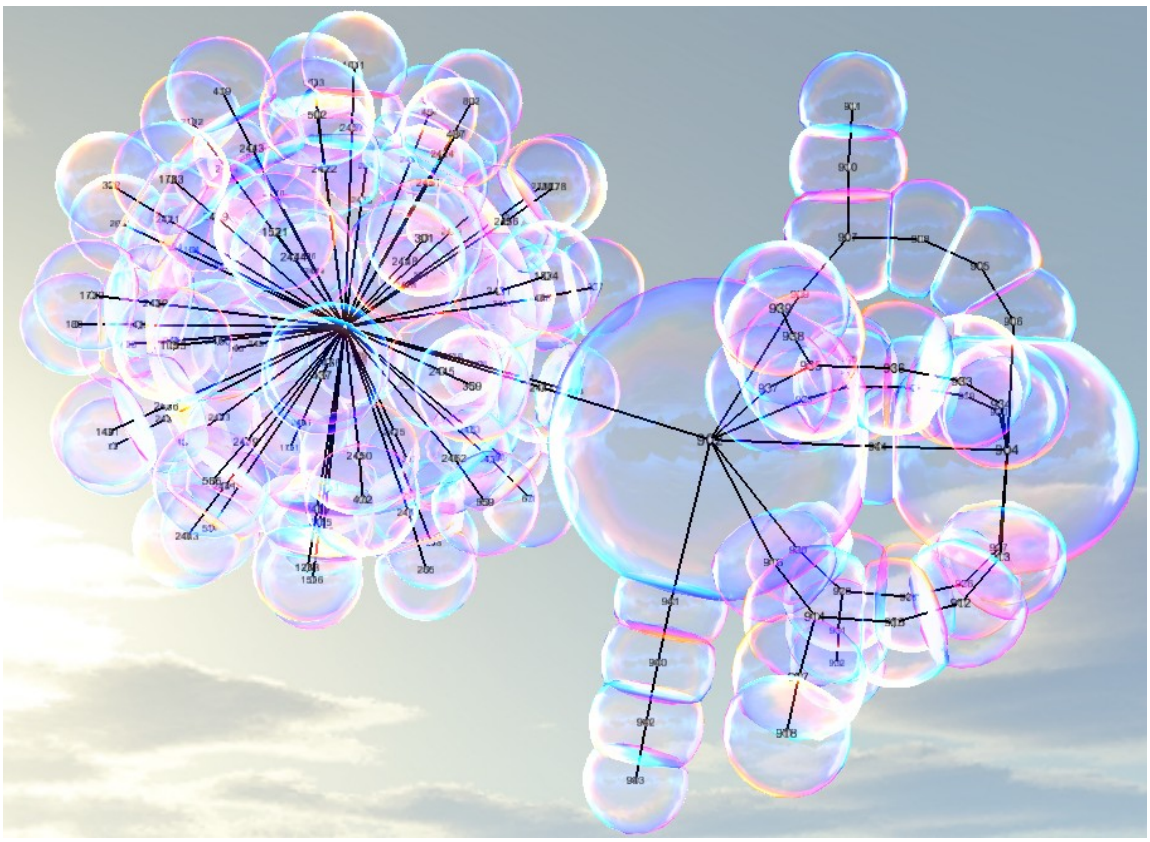


**Figure 11. Software artifacts graph visualized by the soap bubbles metaphor.**

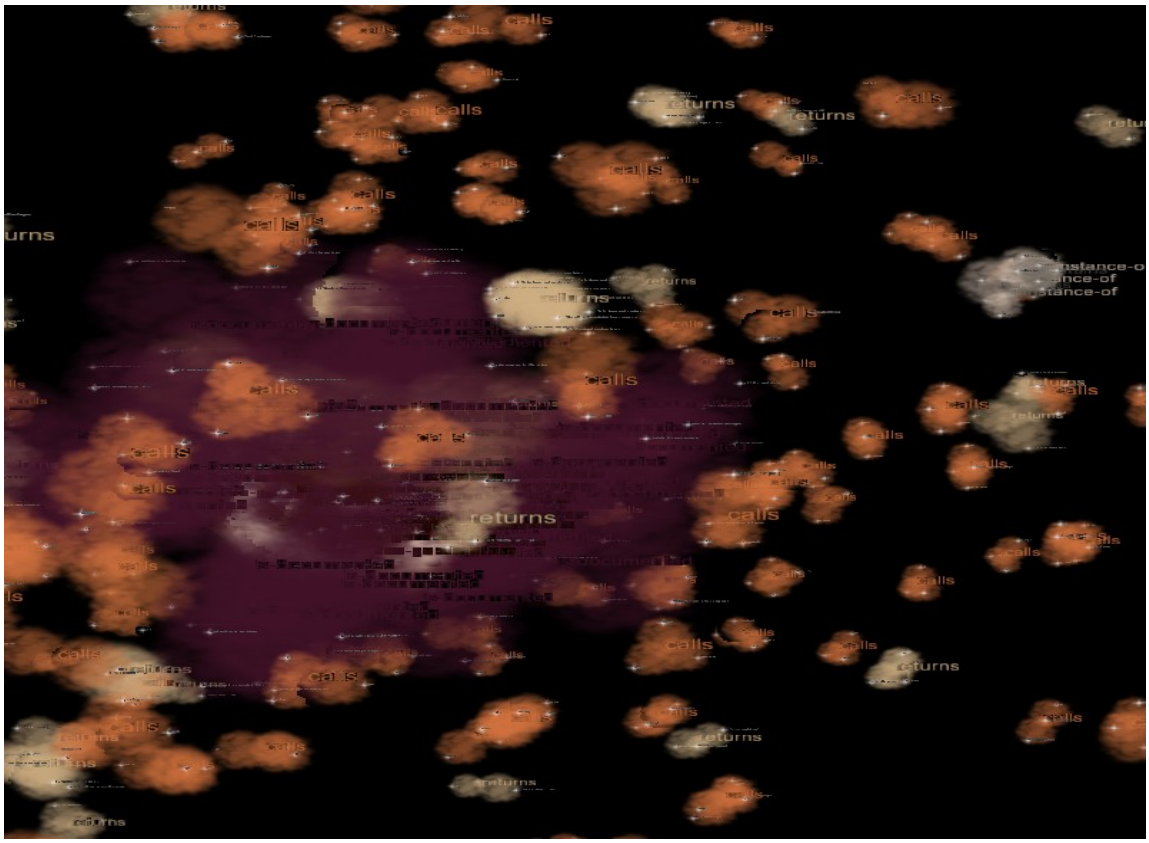



**Figure 12. Software artifacts hypergraph visualized by the nebula metaphor.**

# Reconsidering and Rethinking Quaternionic Special Relativity

Martin Erik Horn

Johann Wolfgang Goethe University
Department of Physics, Institute of Physics Education
Max-von-Laue-Str. 1
D – 60438 Frankfurt on the Main, Germany

m.horn@physik.uni-frankfurt.de

## ABSTRACT

Special relativity can be modelled mathematically with complex quaternions. The relation between this quaternionic special relativity and spacetime algebra will be discussed from a didactical perspective showing the intrinsic relations between quaternion matrices, Pauli matrices, and Dirac matrices.

## Keywords

Special relativity, geometric algebra, Pauli algebra, Dirac algebra, quaternions, physics education.

## 1. INTRODUCTION

After having taught at several schools I returned again to academic life and university some years ago. Knowing nothing about geometric algebra and spacetime algebra at this time one of the ideas I followed was to implement quaternions in spacetime. The didactical path I chose was simple: The Lorentz transformation is a four-dimensional rotation of space and time, and quaternions can be used to describe rotations mathematically in a very elegant manner [Hor02]. Thus I wanted to find a didactical convincing way of modelling Lorentz transformations with quaternions.

Then, years later, I heard about geometric algebra and the rich didactical possibilities it opens. I was immediately overwhelmed by the clear and precise way geometrical meaning is given to algebraic expressions in geometric algebra. And at the same time the profound algebraic meaning appearing behind geometric objects, which can be described algebraically in a clear, precise, and simple way, convinced me [Hor07]. Therefore I nearly at once changed my didactical direction and modelled the Lorentz transformation in spacetime algebra [Hor09], which is the four-dimensional extension of three-dimensional geometric algebra.



It is now time to connect the two loose ends of quaternion algebra and geometric algebra with respect to special relativity and to find a didactical bridge from one system to the other.

## 2. QUATERNIONIC SPECIAL RELATIVITY

Of course I was not the first who tried to formulate special relativistic relations using quaternions. Among others Silberstein [Sil12], [Sil14] Conway [Con47], Lanczos [Lan19], Flint [Fli20] and Blaton [Bla35] were the first who succeeded in doing so. A well-written overview about the historical development of special relativity and the relation to the mathematics of quaternions can be found in Klein (1927/1928) [Kle79], presenting his lectures, which were given before the invention of Pauli matrices in 1927 [Pau27].

In these early quaternionic special relativistic papers every quaternionic basic unit $\vec{i}$, $\vec{j}$, and $\vec{k}$ is then interpreted as basic unit vector in x-, y-, and z-direction. The vector $\vec{r}$ can then be written as linear combination

$$\vec{r} = ct\,\vec{1} + ix\,\vec{i} + iy\,\vec{j} + iz\,\vec{k} \tag{1}$$

with

$$\vec{i}^{\,2} = \vec{j}^{\,2} = \vec{k}^{\,2} = -\vec{1} \tag{2}$$

and

$$\begin{aligned} \vec{i}\,\vec{j} &= -\vec{j}\,\vec{i} = \vec{k} \\ \vec{j}\,\vec{k} &= -\vec{k}\,\vec{j} = \vec{i} \\ \vec{k}\,\vec{i} &= -\vec{i}\,\vec{k} = \vec{j} \end{aligned} \tag{3}$$

The arrows indicate that in this style of writing these objects are indeed thought as vectors [Hor02]. For a

special reason some of the arrows here show to opposite directions.

The basic unit vector $\vec{1}$ with

$$\vec{1}^2 = \vec{1} \tag{4}$$

is interpreted as vector in the direction of time which is commutative with respect to every other mathematical object.

A Lorentz transformation is then normally given as spacetime rotation

$$\vec{r}_{rot} = \vec{q}\ \vec{r}\ \vec{q}^* \tag{5}$$

or $$\vec{r}_{rot} = \vec{q}\ \vec{r}\ \vec{q}^{\sim} \tag{6}$$

with the unit quaternion

$$\vec{q} = q_t\ \vec{1} + iq_x\ \vec{i} + iq_y\ \vec{j} + iq_z\ \vec{k} \tag{7}$$

Which transformation formula (5) resp. (6) should be used depends on how these vectors are defined. They have to mirror the structure of space and time mathematically. Although nature seems to be unique, there are many different mathematical mirrors which can be used to describe our world.

## 3. SPECIAL RELATIVITY IN SPACETIME ALGEBRA

In spacetime algebra the Dirac matrices $\gamma_t$, $\gamma_x$, $\gamma_y$, and $\gamma_z$ with

$$\gamma_t^2 = -\gamma_x^2 = -\gamma_y^2 = -\gamma_z^2 = 1 \tag{8}$$

and
$$\begin{aligned} \gamma_x\,\gamma_y &= -\gamma_y\,\gamma_x & \gamma_t\,\gamma_x &= -\gamma_x\,\gamma_t \\ \gamma_y\,\gamma_z &= -\gamma_z\,\gamma_y & \gamma_t\,\gamma_y &= -\gamma_y\,\gamma_t \\ \gamma_z\,\gamma_x &= -\gamma_x\,\gamma_z & \gamma_t\,\gamma_z &= -\gamma_z\,\gamma_t \end{aligned} \tag{9}$$

are the basic unit vectors of our four-dimensional world we live in [Dor03]. The spacetime vector $\vec{r}$ can then be written as

$$\underline{r} = ct\,\gamma_t + x\,\gamma_x + y\,\gamma_y + z\,\gamma_z \tag{10}$$

and a Lorentz transformation is again a spacetime rotation

$$\underline{r}_{rot} = \underline{\underline{m}}\,\underline{\underline{n}}\,\underline{r}\,\underline{\underline{n}}\,\underline{\underline{m}} \tag{11}$$

now $\underline{n}$ and $\underline{m}$ being two unit reflection vectors

$$\begin{aligned} \underline{n} &= n_t\,\gamma_t + n_x\,\gamma_x + n_y\,\gamma_y + n_z\,\gamma_z \\ \underline{m} &= m_t\,\gamma_t + m_x\,\gamma_x + m_y\,\gamma_y + m_z\,\gamma_z \end{aligned} \tag{12}$$

Because Dirac matrices are behaving like vectors in spacetime algebra, they can be called Dirac vectors.

## 4. TRANSLATING BETWEEN QUATERNION ALGEBRA AND DIRAC ALGEBRA

What is now the relation between these different mathematical structures of quaternion algebra and Dirac algebra which obviously express the same physical situation? To answer this question we have to find a way to translate between these algebras.

Although (2 x 2)-matrices obscure and hide the geometrical meaning of algebraic objects, they can help us to find this translation.

Already before the invention of Pauli matrices it was well known that quaternions can be written as (2 x 2)-matrices [Kle79]. For example they are given in [Bla35, p. 344] as

$$\begin{aligned} \vec{i} &= \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}, \quad \vec{j} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \\ \vec{k} &= \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}, \quad \vec{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned} \tag{13}$$

These matrices are intrinsically connected with Pauli matrices [Pau27]

$$\begin{aligned} \sigma_x &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\ \sigma_z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad 1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned} \tag{14}$$

because the quaternion basic units are mere products of Pauli matrices:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}, \ldots \tag{15}$$

$$\Rightarrow \quad \begin{aligned} \sigma_x\,\sigma_y &= \vec{k} \\ \sigma_y\,\sigma_z &= \vec{i} \\ -\sigma_z\,\sigma_x &= \vec{j} \end{aligned} \tag{16}$$

At the same time Dirac vectors are intrinsically connected with Pauli matrices, because Pauli matrices are mere products of basic Dirac vectors (see for example [Dor03, eq. 5.37]:

$$\Rightarrow \quad \begin{aligned} \gamma_x\,\gamma_t &= \sigma_x \\ \gamma_y\,\gamma_t &= \sigma_y \\ \gamma_z\,\gamma_t &= \sigma_z \end{aligned} \tag{17}$$

Now we are in a position to translate equation (1) into equation (10) and equations (5) or (6) into equation (11).

According to our translation rules (16) the position vector of equation (1)

$$\vec{r} = ct\,\vec{1} + ix\,\vec{i} + iy\,\vec{j} + iz\,\vec{k} \tag{1}$$

is equivalent to the confusing position vector in Pauli algebra

$$\underline{r}' = ct\,1 + ix\,\sigma_y\sigma_z - iy\,\sigma_z\sigma_x + iz\,\sigma_x\sigma_y \tag{18}$$

"Not surprisingly" the minus sign in front of the y-coordinate historically "was a potential source of

great confusion" [Dor03, p. 34] because we have to change from a left-handed coordinate system to a right-handed coordinate system just by flipping the sign of one of the basic quaternion units, e.g.

$$-\vec{j} = \vec{j} = \sigma_z \sigma_x \tag{19}$$

resulting in the right-handed position vector

$$\vec{r} = ct\,\vec{1} + ix\,\vec{i} + iy\,\vec{j} + iz\,\vec{k} \tag{20}$$

and therefore

$$\underline{r} = ct\,1 + ix\,\sigma_y\sigma_z + iy\,\sigma_z\sigma_x + iz\,\sigma_x\sigma_y \tag{21}$$

Inserting the trivector (or pseudoscalar)

$$I = \sigma_x \sigma_y \sigma_z \tag{22}$$

with
$$I^2 = \sigma_x \sigma_y \sigma_z\, \sigma_x \sigma_y \sigma_z = -1 \tag{23}$$

as complex unit $i = I$ into (21) the Pauli position vector transforms into

$$\underline{r} = ct - x\,\sigma_x - y\,\sigma_y - z\,\sigma_z \tag{24}$$

Obviously, this is no pure vector, but a sum of a timelike scalar ct and a one-dimensional spacelike vector part.

This mathematical object of Pauli algebra (with one underlining) can be transformed into an equivalent mathematical object of Dirac algebra (with two underlinings) using equations (17)

$$\underline{\underline{R}} = ct - x\,\gamma_x\gamma_t - y\,\gamma_y\gamma_t - z\,\gamma_z\gamma_t \tag{25}$$

This again is no pure Dirac vector, but a sum of a scalar giving the time coordinate and a Dirac bivector. To finish the translation process, we have to multiply the basic timelike Dirac unit $\gamma_t$ from the left:

$$\begin{aligned}\gamma_t \underline{\underline{R}} &= \gamma_t\, ct - \gamma_t\, x\,\gamma_x\gamma_t - \gamma_t\, y\,\gamma_y\gamma_t - \gamma_t\, z\,\gamma_z\gamma_t \\ &= ct\,\gamma_t + x\,\gamma_x + y\,\gamma_y + z\,\gamma_z = \underline{r}\end{aligned} \tag{26}$$

Now we have got two isomorphic objects: The Dirac multivector $\underline{\underline{R}}$ associated with the quaternion vector $\vec{r}$ has to be multiplied with the timelike basic Dirac unit $\gamma_t$ to get the Dirac vector $\underline{r}$.

We therefore can conclude, that the original quaternion vector geometrically behaves as bivector, that is: as linear combination of oriented area elements. It's sort of a mathematical accident[1] that this Dirac bivector sometimes (e.g. in the case of special relativity) shows the expected Lorentz transformation behaviour of a spacetime vector, which is discussed in the next section.

[1] Freeman Dyson would call it a joke of nature, connected with the strange effects the imaginary unit i can produce [Dys09, p.213].

## 5. REFLECTIONS IN QUATERNION ALGEBRA AND DIRAC ALGEBRA

A reflection at the spacetime unit vector

$$\underline{\underline{n}} = n_t\,\gamma_t + n_x\,\gamma_x + n_y\gamma_y + n_z\,\gamma_z \tag{27}$$

is given in Dirac algebra by

$$\underline{r}_{ref} = \pm\, \underline{\underline{n}}\, \underline{r}\, \underline{\underline{n}} \tag{28}$$

while the positive sign is used when the reflection vector $\underline{\underline{n}}$ is a timelike vector $\underline{\underline{n}}^2 = 1$ and the negative sign is used when the reflection vector $\underline{\underline{n}}$ is a spacelike vector $\underline{\underline{n}}^2 = -1$.

Let's try to convert this back into quaternion algebra.

$$\begin{aligned}\gamma_t\, \underline{r}_{ref} &= \pm\, \gamma_t\, \underline{\underline{n}}\, \underline{r}\, \underline{\underline{n}} \\ &= \pm\, \gamma_t\, \underline{\underline{n}}\, \underline{r}\, \gamma_t\, \gamma_t\, \underline{\underline{n}}\end{aligned} \tag{29}$$

with

$$\begin{aligned}\underline{\underline{N}} &= \gamma_t \underline{\underline{n}} = n_t - n_x\,\gamma_x\gamma_t - n_y\,\gamma_y\gamma_t - n_z\,\gamma_z\gamma_t \\ \underline{\underline{R}}' &= \underline{r}\,\gamma_t = ct + x\,\gamma_x\gamma_t + y\,\gamma_y\gamma_t + z\,\gamma_z\gamma_t\end{aligned} \tag{30}$$

Please compare the different signs of $\underline{\underline{R}}'$ and $\underline{\underline{R}}$ in equations (25) and (30). Converting back into Pauli algebra (with one underlining) gives

$$\underline{r}_{ref} = \pm\, \underline{\underline{n}}\, \underline{r}'\, \underline{\underline{n}} \tag{31}$$

with

$$\begin{aligned}\underline{\underline{n}} &= n_t - n_x\,\sigma_x - n_y\,\sigma_y - n_z\,\sigma_z \\ \underline{r}' &= ct + x\,\sigma_x + y\,\sigma_y + z\,\sigma_z\end{aligned} \tag{32}$$

Substituting

$$\begin{aligned}i\,\sigma_y\sigma_z &= -\sigma_x \\ i\,\sigma_z\sigma_x &= -\sigma_y \\ i\,\sigma_x\sigma_y &= -\sigma_z\end{aligned} \tag{33}$$

changes equations (32) into

$$\begin{aligned}\underline{\underline{n}} &= n_t\,1 + in_x\,\sigma_y\sigma_z + in_y\,\sigma_z\sigma_x + in_z\,\sigma_x\sigma_y \\ \underline{r}' &= ct\,1 - ix\,\sigma_y\sigma_z - iy\,\sigma_z\sigma_x - iz\,\sigma_x\sigma_y\end{aligned} \tag{34}$$

Now (31) can be translated back into quaternion algebra:

$$\vec{r}_{ref}\,' = \pm\, \vec{n}\, \vec{r}\,'\, \vec{n} \tag{35}$$

with

$$\begin{aligned}\vec{n} &= n_t\,\vec{1} + in_x\,\vec{i} + in_y\,\vec{j} + in_z\,\vec{k} \\ \vec{r}_{ref}\,' &= ct\,\vec{1} - ix\,\vec{i} - iy\,\vec{j} - iz\,\vec{k}\end{aligned} \tag{36}$$

To get rid of the minus signs in (30) we have to change equation (35) into

$$\vec{r}_{ref} = \pm\, \vec{n}\, \vec{r}^{\,*}\, \vec{n} \tag{37}$$

in a right-handed coordinate system. Thus it is possible to express spacetime reflections in quaternion algebra, but it is of course not as elegant as in spacetime algebra.

## Examples

To illustrate the possible different cases three examples are shown below. First a pure space reflection of the spacetime vector

$$\vec{r} = ct\,\vec{1} + ix\,\vec{i} + iy\,\vec{j} + iz\,\vec{k}$$

or $\underline{r} = ct\,\gamma_t + x\,\gamma_x + y\,\gamma_y + z\,\gamma_z$

at the x-axes with reflection vector

$$\vec{n}_1 = i\,\vec{i} \qquad \text{with} \quad \vec{n}_1\,\vec{n}_1{}^* = -1$$

or $\underline{n}_1 = \gamma_x \qquad \text{with} \quad \underline{n}_1{}^2 = -1$

gives the reflected vector

$$\vec{r}_{1ref} = -i\vec{i}\;\vec{r}\,^*\,i\vec{i}$$
$$= -ct\,\vec{1} + ix\,\vec{i} - iy\,\vec{j} - iz\,\vec{k}$$

or $\underline{r}_{1ref} = -\gamma_x\,(ct\,\gamma_t + x\,\gamma_x + y\,\gamma_y + z\,\gamma_z)\,\gamma_x$
$= -ct\,\gamma_t + x\,\gamma_x - y\,\gamma_y - z\,\gamma_z$

with all components exchanging the sign except the x-coordinate.

Secondly, a pure space reflection of the spacetime vector $\vec{r}$ or $\underline{r}$ at the diagonal line of the xy-plane with reflection vector

$$\vec{n}_2 = \frac{1}{\sqrt{2}}\left(i\,\vec{i} + i\,\vec{j}\right) \qquad \text{with} \quad \vec{n}_2\,\vec{n}_2{}^* = -1$$

or $\underline{n}_2 = \frac{1}{\sqrt{2}}\left(\gamma_x + \gamma_y\right) \qquad \text{with} \quad \underline{n}_2{}^2 = -1$

gives the reflected vector

$$\vec{r}_{2ref} = -\frac{1}{2}\left(i\,\vec{i} + i\,\vec{j}\right)\,\vec{r}\,^*\,\left(i\,\vec{i} + i\,\vec{j}\right)$$
$$= -ct\,\vec{1} + iy\,\vec{i} + ix\,\vec{j} - iz\,\vec{k}$$

or $\underline{r}_{2ref} = -\frac{1}{2}(\gamma_x + \gamma_y)(ct\,\gamma_t + x\,\gamma_x + y\,\gamma_y + z\,\gamma_z)(\gamma_x + \gamma_y)$
$= -ct\,\gamma_t + y\,\gamma_x + x\,\gamma_y - z\,\gamma_z$

with negative components of time- and the z-coordinate and the components of the x- and y-directions exchanged.

As third example the spacetime reflection of the spacetime vector $\vec{r}$ or $\underline{r}$ at the time-axes with reflection vector

$$\vec{n}_3 = \vec{1} \qquad \text{with} \quad \vec{n}_3\,\vec{n}_3{}^* = +1$$

or $\underline{n}_1 = \gamma_t \qquad \text{with} \quad \underline{n}_3{}^2 = +1$

is considered. Now the reflected vector

$$\vec{r}_{3ref} = \vec{1}\;\vec{r}\,^*\;\vec{1}$$
$$= ct\,\vec{1} - ix\,\vec{i} - iy\,\vec{j} - iz\,\vec{k}$$

or $\underline{r}_{3ref} = \gamma_t\,(ct\,\gamma_t + x\,\gamma_x + y\,\gamma_y + z\,\gamma_z)\,\gamma_t$
$= ct\,\gamma_t - x\,\gamma_x - y\,\gamma_y - z\,\gamma_z$

arises, and all components of space directions exchange the sign while the component of the time-coordinate does not change.

Finally the spacetime reflection of the spacetime vector $\vec{r}$ or $\underline{r}$ at the timelike unit reflection vector

$$\vec{n}_4 = \frac{1}{4}\left(5\cdot\vec{1} + 3i\,\vec{i}\right) \qquad \text{with} \quad \vec{n}_4\,\vec{n}_4{}^* = +1$$

or $\underline{n}_4 = \frac{1}{4}(5\,\gamma_t + 3\,\gamma_x) \qquad \text{with} \quad \underline{n}_4{}^2 = +1$

gives the reflected vector

$$\vec{r}_{4ref} = \frac{1}{16}\left(5\cdot\vec{1} + 3i\,\vec{i}\right)\,\vec{r}\,^*\,\left(5\cdot\vec{1} + 3i\,\vec{i}\right)$$
$$= \frac{34ct - 30x}{16}\,\vec{1} + i\frac{30ct - 34x}{16}\,\vec{i} - iy\,\vec{j} - iz\,\vec{k}$$

or $\underline{r}_{4ref} = \frac{1}{16}(5\gamma_t + 3\gamma_x)(ct\gamma_t + x\gamma_x + y\gamma_y + z\gamma_z)(5\gamma_t + 3\gamma_x)$

$$= \frac{34\,ct - 30\,x}{16}\,\gamma_t + \frac{30\,ct - 34\,x}{16}\,\gamma_x - y\,\gamma_y - z\,\gamma_z$$
$$= (2{,}125\,ct - 1{,}875\,x)\,\gamma_t + (1{,}875\,ct - 2{,}125\,x)\,\gamma_x - y\,\gamma_y - z\,\gamma_z$$

which can be checked by squaring:

$$\vec{r}_{4ref}\,\vec{r}_{4ref}{}^* = \vec{r}\,\vec{r}^* = (ct)^2 - x^2 - y^2 - z^2$$
$$\underline{r}_{4ref}{}^2 = \underline{r}^2 = ct^2 - x^2 - y^2 - z^2$$

## 6. ROTATIONS IN QUATERNION ALGEBRA AND DIRAC ALGEBRA

A succession of two reflections gives always a rotation, and every rotation can be decomposed into two (or four or six or any other even number of) reflections.

Therefore a reflection of the reflected Dirac vector $\underline{r}_{ref}$ at the spacetime unit vector

$$\underline{m} = m_t\,\gamma_t + m_x\,\gamma_x + m_y\,\gamma_y + m_z\,\gamma_z \qquad (38)$$

is given in Dirac algebra by

$$\underline{r}_{rot} = \pm\,\underline{m}\,\underline{r}_{ref}\,\underline{m} \qquad (39)$$

and leads to a spacetime rotation. As usual the positive sign is used when the reflection vector $\underline{m}$ is a timelike vector $\underline{m}^2 = 1$ and the negative sign is used when the reflection vector $\underline{m}$ is a spacelike vector $\underline{m}^2 = -1$.

When both reflection vectors are of equal quality (both spacelike or both timelike) the signs cancel

$$\underline{r}_{rot} = \pm\,\underline{m}\,(\pm\,\underline{n}\,\underline{r}\,\underline{n})\,\underline{m}$$
$$= \underline{m}\,\underline{n}\,\underline{r}\,\underline{n}\,\underline{m} \qquad (40)$$

Let's try again to convert this back into quaternion algebra.

$$\gamma_t\,\underline{r}_{rot} = \gamma_t\,\underline{m}\,\underline{n}\,\underline{r}\,\underline{n}\,\underline{m}$$
$$= \gamma_t\,\underline{m}\,\underline{n}\,\gamma_t\,\gamma_t\,\underline{r}\,\underline{m}\,\gamma_t\,\gamma_t\,\underline{n} \qquad (41)$$

This time we do not have to care about the signs of the original vector $\underline{R} = \gamma_t\,\underline{r}$, but of the signs of the unit rotation quaternions, as two successive applications of equation (37) show:

$$\vec{r}_{rot} = \pm\, \vec{m}\, \vec{r}_{ref}\, {}^{*}\, \vec{m}$$
$$= \pm\, \vec{m}\, (\pm \vec{n}\, \vec{r}\, {}^{*}\, \vec{n})\, {}^{*}\, \vec{m} \qquad (42)$$
$$= \vec{m}\, \vec{n}\, {}^{*}\, \vec{r}\, \vec{n}\, {}^{*}\, \vec{m}$$

Consistently we have to use equation (6), and the unit rotation quaternions become

$$\vec{q} = \vec{m}\, \vec{n}\, {}^{*}$$

and $$\vec{q}\tilde{} = \vec{n}\, {}^{*}\, \vec{m} \qquad (43)$$

in a right-handed coordinate system.

## Examples

The differences and similarities between quaternionic rotations and Dirac spacetime rotations will be illustrated in the following. First we reflect the time-like Point A (symbolised by the vector $\vec{r}_A$ or $\underline{r}_A$, which lies inside the future light cone of an observer at the origin of the coordinate system) and the light-like point B (symbolised by the vector $\vec{r}_B$ or $\underline{r}_B$, which lies on the future light cone)

$$\vec{r}_A = 5 \cdot \vec{1} + 4i\, \vec{i}$$
$$\vec{r}_B = 4 \cdot \vec{1} + 4i\, \vec{i}$$

or $$\underline{r}_A = 5\, \gamma_t + 4\, \gamma_x$$
$$\underline{r}_B = 4\, \gamma_t + 4\, \gamma_x$$

at the unit reflection vectors

$$\vec{n} = \frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} + i\, \vec{i}\right)$$
$$\vec{m} = \vec{1}$$

or $$\underline{n} = \frac{1}{\sqrt{8}}(3\, \gamma_t + \gamma_x)$$
$$\underline{m} = \gamma_t$$

with $$\vec{n}\, \vec{n}^{*} = \vec{m}\, \vec{m}^{*} = +1$$
$$\underline{n}^2 = \underline{m}^2 = +1$$

The reflected position vectors then are

$$\vec{r}_{Aref} = 3{,}25 \cdot \vec{1} - 1{,}25i\, \vec{i}$$
$$\vec{r}_{Bref} = 2 \cdot \vec{1} - 2i\, \vec{i}$$

or $$\underline{r}_{Aref} = 3{,}25\, \gamma_t - 1{,}25\, \gamma_x$$
$$\underline{r}_{Bref} = 2\, \gamma_t - 2\, \gamma_x$$

With $$\vec{q} = \frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} - i\, \vec{i}\right)$$ and

$$\vec{q}\tilde{} = \frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} - i\, \vec{i}\right)$$

the rotated Dirac position vectors become

$$\vec{r}_{Arot} = \frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} - i\vec{i}\right)\left(5 \cdot \vec{1} + 4i\, \vec{i}\right)\frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} - i\vec{i}\right)$$
$$= \frac{1}{8}\left(11 \cdot \vec{1} + 7i\vec{i}\right)\left(3 \cdot \vec{1} - i\vec{i}\right)$$
$$= 3{,}25 \cdot \vec{1} + 1{,}25i\vec{i}$$

$$\vec{r}_{Brot} = \frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} - i\vec{i}\right)\left(4 \cdot \vec{1} + 4i\, \vec{i}\right)\frac{1}{\sqrt{8}}\left(3 \cdot \vec{1} - i\vec{i}\right)$$
$$= \frac{1}{8}\left(8 \cdot \vec{1} + 8i\vec{i}\right)\left(3 \cdot \vec{1} - i\vec{i}\right)$$
$$= 2 \cdot \vec{1} + 2i\vec{i}$$

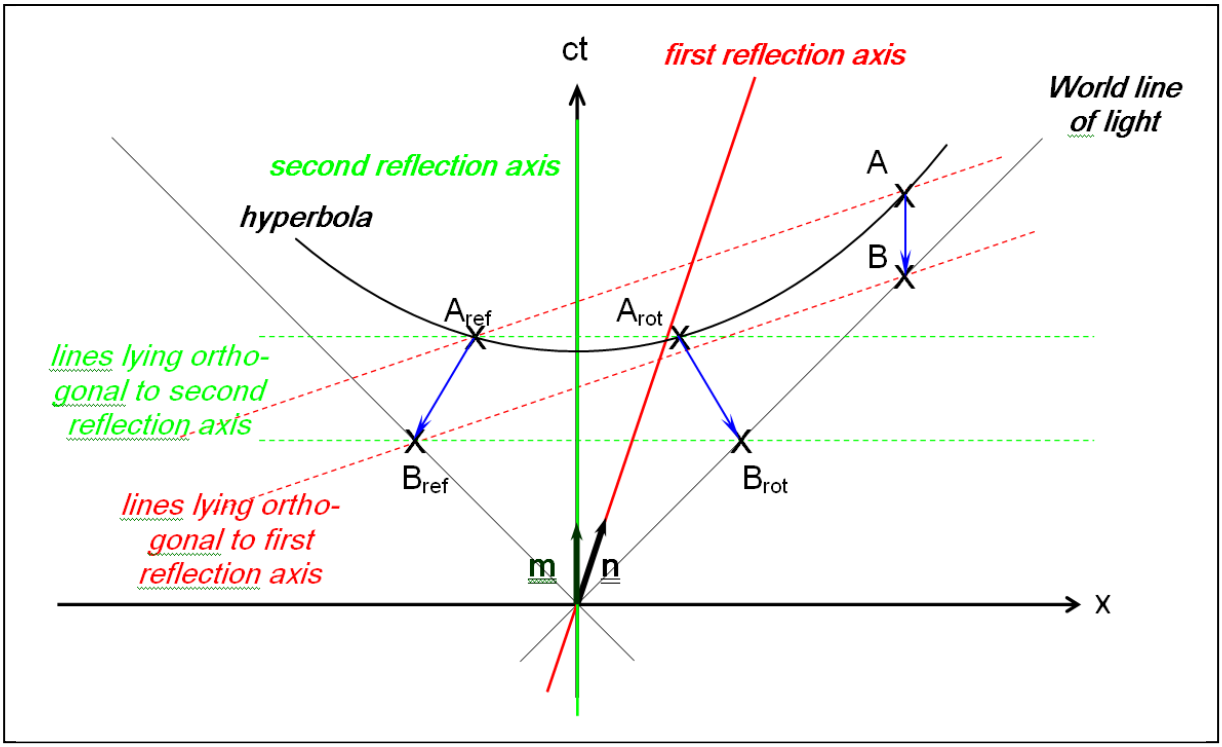



**Figure 1. Construction of a spacetime rotation of two points A and B by two successive spacetime reflections at the first (red) and second (green) reflection axes.**

or $\underline{r}_{=Arot} = \frac{1}{\sqrt{8}}(3-\gamma_x\gamma_t)(5\gamma_t+4\gamma_x)\frac{1}{\sqrt{8}}(3+\gamma_x\gamma_t)$

$= \frac{1}{8}(11\gamma_t+7\gamma_x)(3+\gamma_x\gamma_t)$

$= 3{,}25\ \gamma_t + 1{,}25\ \gamma_x$

$\underline{r}_{=Brot} = \frac{1}{\sqrt{8}}(3-\gamma_x\gamma_t)(4\gamma_t+4\gamma_x)\frac{1}{\sqrt{8}}(3+\gamma_x\gamma_t)$

$= \frac{1}{8}(8\gamma_t+8\gamma_x)(3+\gamma_x\gamma_t)$

$= 2\ \gamma_t + 2\ \gamma_x$

(see figure 1). The original and the transformed points B always lie on the world line of light, as lightlike vectors remain lightlike when reflected or rotated.

Thus both mathematical concepts describe Lorentz transformation appropriate and correct, and uncover the geometrical meaning of this transformation.

## 7. METACONCEPTUAL AWARENESS IN PHYSICS AND MATHEMATICS

Quaternions played a major role in the historical development of mathematics and physics. When lecturing about this development Felix Klein emphasized the importance of Hamilton's and Grassmann's ideas several times [Kle79]. Analyzing the heuristic role of quaternions Anderson and Joshi thus concluded, that "unique features of quaternionic structures have been woven closely into the development of new physical theories" [And02, p. 15].

These intrinsic relationship was already discussed by Grassmann [Gra77], who connected his theory of extensions with the theory of quaternions formulated nearly at the same time by Hamilton. Quaternions and geometric algebra give us two different mathematical views on our world. Obviously this world outside us is (as far as we know) unique, one and single, but we do not have a unique and single way to describe this world mathematically.

We posses lots of different mathematical languages (or as Hestenes once said: "a Babel of mathematical tongues" [Hes03, p. 106]). But we do not know which mathematical language will be appropriate to solve the physics problems of the future. We only know which language today fits best to solve our contemporary problems. Therefore it is our task not only to teach and explain geometric or spacetime algebra, but to present and discuss the position geometric algebra takes by exploring the relation to other and different mathematical concepts. Let it never happen again that a prominent scientist has to say with respect to the relation between the Dirac equation and geometry: "Had we been better educated in physics, or had there been the kind of dialogue with physicists that is now common, we would have got there much sooner" [Ati08, p. 116].

For this reason we should pay attention to developing meta-conceptual awareness when teaching physics and mathematics or other fields of science. Our students should be able to change mathematical and physical perspectives to look at problems from different angles and to analyze and explain problems in distinct ways.

As a research scientist it might be frustrating that "much time (is) being taken up with mere translation between the two modes of expression" [Hes71, p. 1013]. But as a teacher or lecturer of physics it is necessary to push and to urge our students to think through these translations. One of the most promising ways to implement (that is to learn) knew knowledge is to create cognitive conflicts between different perspectives to let the students find a solution of these conflicts. We would be bad teachers of physics and mathematics if we presented one and only one truth.

As a final remark I want to look back into the history of mathematics and especially into the history of the quaternionists who wanted to promote quaternion algebra – and who failed. We can learn something about this failure. Felix Klein and Arnold Sommerfeld used the theory of quaternions in a convincing and impressive way to explain the physics of the gyroscope (see section 7 of chapter I in [Kle97]). At the end of this chapter Klein and Sommerfeld wrote: "...we want to bring forward also an advantage which is as well attributed to the theory of quaternions and vector analysis, namely the independence of their operations and their basic units from the coordinate system. However ... it would mean to misunderstand the character of analytic geometry if we always and in principle would not use coordinates explicitly when performing calculations." [Kle97, p.68].

Klein and Sommerfeld here describe the struggle between supporters of coordinate free methods in quaternion algebra and the supporters of a more frequent use of coordinates. Their conclusion: "It is important to think invariantly, not to calculate invariantly."

In geometric algebra we are in a similar situation today, when supporters of coordinate free methods claim that essential calculation should be done without coordinates. Is this really possible? Goldman gives a clear answer: "Does the full geometric algebra really lead to coordinate free methods for all of Computer Graphics? Again in my experience the answer is no" [Gol08, p. 656].

We should therefore show a meta-conceptual openness and teach both strategies: working with and working without coordinates. The invention of coordinates was at least a decisive step in the conceptual development of modern mathematics, as Freeman

Dyson writes: “It means that the deepest concepts in mathematics are those which link one world of ideas with another. In the seventeenth century Descartes linked the disparate worlds of algebra and geometry with his concept of coordinates” [Dys09, p. 218]. In a similar way geometric algebra works: It again links the disparate worlds of algebra and geometry.

## 8. REFERENCES


[And02] Anderson, R. & Joshi, G.C.: Quaternions and the Heuristic Role of Mathematical Structures in Physics, arxiv:hep-ph/9208222v2 (4. Sept. 2002), http://arxiv.org/abs/hep-ph/9208222.

[Ati98] Atiyah, M.F.: The Dirac equation and geometry. Published in: Goddard, P. (Ed.): Paul Dirac – The Man and His Work, pp. 108 - 124, Cambridge University Press, Cambridge 1998.

[Bay04] Baylis, W.E. & Sobczyk, G.: Relativity in Clifford’s Geometric Algebras of Space and Spacetime, International Journal of Theoretical Physics, No. 10, Vol. 43 (2004), pp. 2061 - 2079.

[Bla35] Blaton, J.: Quaternionen, Semivektoren und Spinoren, Zeitschrift für Physik, No. 5/6, Vol. 95 (1935), pp. 337 - 354.

[Con47] Conway, A.W.: Applications of Quaternions to Rotations in Hyperbolic Space of Four Dimensions, Proceedings of the Royal Society of London, Series A, No. 1025, Vol. 191 (1947), pp. 137 - 145.

[Dor03] Doran Ch. & Lasenby, A.: Geometric Algebra for Physicists, Cambridge University Press, Cambridge 2003.

[Dys09] Dyson, F.: Birds and Frogs, AMS Einstein Lecture, Notices of the American Mathematical Society, No. 2, Vol. 56 (2009), pp. 212 - 223.

[Fli20] Flint, H.T.: XLIII. Applications of quaternions to the theory of relativity, Philosophical Magazine Series 6, No. 232, Vol. 39 (1920), pp. 434 - 449.

[Gol08] Goldman, R.: After the revolution: Geometric algebra for Computer Scientists in the twenty-first century, book review, Computer-Aided Design, Vol. 40 (2008), pp. 655 - 656.

[Gra77] Grassmann, H.G.: Über den Ort der Hamilton’schen Quaternionen in der Ausdehnungslehre, Mathematische Annalen, Vol. 12 (1877), pp. 375 - 386.

[Hes71] Hestenes, D.: Vectors, Spinors, and Complex Numbers in Classical and Quantum Physics, American Journal of Physics, No. 9, Vol. 39 (1971), pp. 1013 - 1027.

[Hes03] Hestenes, D.: Reforming the Mathematical Language of Physics, Oersted Medal Lecture, American Journal of Physics, No. 2, Vol. 71 (2003), pp. 104 -121.

[Hor02] Horn, M.E.: Quaternionen in der Hochschulphysik am Beispiel der Speziellen Relativitätstheorie. Published in: Nordmeier, V. (Red.): Tagungs-CD der DPG-Frühjahrstagung des Fachverbands Didaktik der Physik in Leipzig, Beitrag 26.24, LOB – Lehmanns Media, Berlin 2002. See also: Quaternions in University-Level Physics Considering Special Relativity, arxiv:physics/0308017v1 (5. Aug. 2003), http://arxiv.org/abs/physics/0308017.

[Hor07] Horn, M.E.: Quaternionen und Geometrische Algebra. Published in: Nordmeier, V. & Oberländer, A. (Eds.). Tagungs-CD der DPG-Frühjahrstagung des Fachverbands Didaktik der Physik in Kassel, Beitrag 28.2, LOB – Lehmanns Media, Berlin 2006. See also: Quaternions and Geometric Algebra, arxiv:physics/0709.2238v1 (14. Sept. 2007), http://arxiv.org/abs/0709.2238.

[Hor09] Horn, M.E.: Vom Raum zur Raumzeit. Published in: Höttecke, D. (Ed.): Chemie- und Physikdidaktik für die Lehramtsausbildung, Beiträge zur Jahrestagung der GDCP in Schwäbisch Gmünd, Band 29, pp. 455 - 457, LIT-Verlag Dr. W. Hopf, Berlin 2009.

[Kle97] Klein, F. & Sommerfeld, A.: Über die Theorie des Kreisels, Heft I, B. G. Teubner Verlag, Leipzig 1897.

[Kle79] Klein, F.: Vorlesungen über die Entwicklung der Mathematik im 19. Jahrhundert. Part I published as Vol. 24 and part II as Vol. 25 of Courant, R. (Ed.): Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete, Julius Springer Verlag, Berlin 1926 and 1927. Reprint: Springer Verlag Berlin, Heidelberg, New York 1979.

[Lan19] Lanczos, C.: Die Funktionentheoretischen Beziehungen der Maxwellschen Äthergleichungen. Ein Beitrag zur Relativitäts- und Elektronentheorie. Dissertation, Universität Budapest, Verlagsbuchhandlung Josef Németh, Budapest 1919. See also: The Functional Theoretical Relationships of the Homogenous Maxwell Equations, arxiv:physics/04080079v1 (17. Aug. 2004), http://arxiv.org/abs/physics/0408079.

[Pau27] Pauli, W.: Zur Quantenmechanik des magnetischen Elektrons, Zeitschrift für Physik, No. 9/10, Vol. 43 (1927), pp. 601 - 623.

[Sil12] Silberstein, L.: LXXVI. Quaternionic form of relativity, Philosophical Magazine Series 6, No. 137, Vol. 32 (1912), pp. 790 - 809.

[Sil14] Silberstein, L.: The Theory of Relativity, Macmillan & Co., London 1914.

# Impact Crater Detection on Mars Digital Elevation and Image Model

Mert Degirmenci
Middle East Technical University,
Turkey
mert.degirmenci@ceng.metu.edu.tr

Shatlyk Ashyralyev
Middle East Technical University,
Turkey
shatlyk.ashyralyyev@ceng.metu.edu.tr

**ABSTRACT**

As outer space image acquisition techniques progress, larger amounts of planetary data sets become available. Impact crater statistics about planets is an important resource as use of this information reveals geological history. Since manual detection of impact craters requires substantial human resource, there is a compelling need to investigate automated crater detection algorithms. In this study, we develop a novel framework to detect Martian impact craters by fusing data obtained from Mars Global Surveyor. In our proposed method, extracted craters from Mars Digital Image Model (MDIM) are crosschecked by using Mars Digital Elevation Model (MDEM). Multi population genetic algorithm (MPGA) has been devised to extract craters from scale invariant feature set found by SIFT algorithm. In order to decrease the number of false positives, extracted from MDIM are validated by detected basins from MDEM. Experimental results on NASA databases suggest high crater detection rates.

## 1 INTRODUCTION

Impact craters are formed by collision of two celestial bodies. Planetary science utilizes the impact crater databases to extract characteristic information about both colliding bodies. Reliably extracted crater features enable geologists inspect hydrological processes, and climatic information about the planet under consideration. Surface age prediction also relies on size and frequency distributions of craters. Quest for geological information about recently scanned planets stimulates the need for impact crater databases. Impact crater detectors are also utilized in space exploration. Spacecrafts need to incorporate a crater detector for visual positioning. In order to land on asteroids autonomously, spacecrafts must calculate the locations of the impact craters based on 3D model of the space and 2D images obtained. Both usage area of a crater detector system requires highly accurate results.

In order to create credible crater databases, a number of scientists have manually examined the optical satellite images and gathered information about crater features. The most comprehensive data set, known as Barlow catalog, includes characteristic information for more than 40,000 craters on Mars [Bar88]. Even though visual inspection of satellite images may reveal key information about impact craters, this process eventually becomes infeasible upon arrival of large volume sensor data. Recently acquired sensor data have increased the number of research studies about automatic impact crater detection.

Significant number of researchers has used optical images to detect impact craters. These visibility based methods have limitations with regard to illumination, surface characteristics, and occlusion. Although most impact craters have obvious circular features, impact angle and geological deformations severely affect visibility of them. Significant overlap between craters also degrades accuracy of automatic crater detectors using optical sensor data captured at frequently hit areas of the planets. To address these challenges, we propose a data fusion approach for impact crater detection. Our algorithm reduces the error by fusing elevation data and optical data. In this section, we address previously researched optical image-based and elevation-based crater detection algorithms.

Existing body of research on crater detection algorithms generally focus on optical images to produce scalable crater databases. Most of the proposed frameworks incorporate either unsupervised or supervised methods to identify features and whereabouts of impact craters. Unsupervised techniques focus on finding rims and merging them to locate the crater. Hough transform based methods are generally incorporated in this class of techniques. Supervised learning methods, on the other hand, involve kernel-based and neural network based learning methods for training. Support Vector Machines are usually used as classifiers in crater detection.

Since high level of accuracy is needed for a crater database to be utilized by planetary scientists, researchers have combined several crater detection algorithms in order to produce more accurate results. Sawabe et al. have used multiple boundary based approaches and merged the results obtained [Saw06]. In the first approach they have used images that are classified considering illumination. When shady and illuminated pattern is recognized, they fit a circle to the surrounding edges. Although abrupt brightness changes may reveal a lot about the surface under consideration, the presence of sensor data with correct illumination is often an unrealistic assumption to make. The second approach they have used tries to find edge pixels of interest using a vectorized feature extractor proposed by Sugiyama et al. [Sug97]. Then a roundness measure is checked for identification of circles. Other two approaches proposed by Sawabe et al. uses Hilditch's thinning algorithm and fuzzy Hough transform in addition to previously discussed algorithms respectively.

All of the described approaches proposed by Sawabe et al. up to now suffer from elliptic shape of impact craters. Depending on geological deformations on the sur-

face, there is a high possibility that craters form degraded ellipses rather than circles on the surface. In fact, most craters in the Barlow catalog can hardly be characterized by circularity features [Bar88].

Machine learning approaches have also been applied in order to detect and catalog impact craters. Wetzler et al. have used various supervised learning algorithms, including ensemble methods (bagging and AdaBoost with feed-forward neural networks as base learners), support vector machines (SVM), and continuously scalable template models (CSTM) to derive crater detectors from ground-truthed images [Wet05]. They have noted that the SVM solution to the problem performs superior on crater detection and localization compared to boundary-based approaches such as Hough Transform [Wet05]. However, their implementation demands huge ground-truth data and computational resource considering that SVM models they found involved approximately five thousand support vectors. In order to overcome large computational demands, they have proposed using blocked-FFT implementation of the SVM decision function [Bur04].

A number of researchers have used combination of supervised and unsupervised techniques to detect impact craters. Kim et al. propose three staged crater detection system [Kim05]. In the first stage, they eliminate noise in the image by extracting region of interest. They also consider edge direction, and illumination angle at this stage. In the second stage of their algorithm, which they call organization stage, primitive arcs are organized by graph and conic section fitting. Candidate craters are propagated to the last stage, where they are verified by a fitness measure and a false crater classifier based on artificial neural networks.

Honda et al. have also combined machine learning approaches with boundary based methods [Hon00]. In the framework they have proposed, image is first binarized. To find craters, circular object detection is then applied using a combination of Hough Transform and Genetic Algorithm. At the last stage, they have utilized Self-Organizing Maps to categorize candidate craters. The two frameworks discussed above incorporate both supervised learning techniques and boundary based analysis of optical satellite imagery.

Ellipse fitting algorithms are frequently used when researchers model the crater to be detected as an ellipse. Clustering techniques such as K-means are generally used for partitioning the feature points into set of candidate craters. Leroy et al. have employed the same idea to isolate individual craters [Ler01]. After partitioning, they have fit an ellipse on the boundary of the craters. Although boundary based methods provide simple yet powerful crater detectors, they noted that illumination angle and noisy sensor data may obstruct detecting impact craters. In order to alleviate these problems, researchers have complicated the focusing process of crater detection by smoothing and applying morphological operations to optical images. Marchetti et al. notes that smoothing image increases robustness to noise significantly [Mar04]. However, the information contained in optical data for overlapping craters may be lost due to smoothed image.

Recently, NASA revealed sufficiently precise and comprehensive digital image (DIM), and elevation (DEM) data on Mars. This advancement lead to more reliable crater extractors. Researchers have used raw DEM data to detect impact craters. Bue et al. have discussed limitations of optical image data and outlined an algorithm using solely digital elevation model [Bue07]. They utilized elevation of the surface to detect basins. Idea of Bue's study was to merge high curvature edges and basins of elevation model to detect crater rims. Located crater rims are passed through a set of morphological operations to thin and close the gaps. They applied Hough Transform to detect craters and noted significant improvements over optical image based crater detectors. Their findings are important to us since this was the first study using DEM to detect impact craters. Improvements can be promised over their implementation by incorporating digital image data. Machine learning approaches can also be included to increase the accuracy of their craters detector.

In this study, we address inherent challenges in crater detection such as limitations of image acquisition techniques and deformations of craters. We improve the accuracy of existing crater detectors by fusing the results obtained from height data and optical image data. Next section gives an overview of the framework proposed.

## 2 OVERVIEW

Our framework can be decomposed into two modules. These are ellipse detection and basin detection modules. The two result set obtained are merged at the end to increase the reliability of the algorithm. In this section, algorithms involved in both components of the system are described. Following sections include more detailed discussion of the methods.

Optical image processing module first computes scale invariant feature transform of the image proposed by Lowe [Low99]. Main reason we have used SIFT features is their robustness to scale, orientation, and affine distortion. Scale invariance is especially important considering high scale variance between craters to be detected on Mars. These features fed into a multi population genetic algorithm to find ellipses. Detected ellipses are verified by results of DEM processing module.

Elevation data processing module smoothes the height map of the Mars surface. Smoothing the surface increases the accuracy of the basin extraction process. The basins are found using drainage network extraction algorithm proposed by Freeman et al. [Fre92]. Sink sources of the height map are generally craters to be detected. However, Martian landscape involves non-crater basins as well. Thus, basin detection module of the framework is generally not enough to be used as a reliable crater extractor. This is the reason we fuse the results obtained of basin and crater detection modules. The flowchart of the system is given in Figure (1).

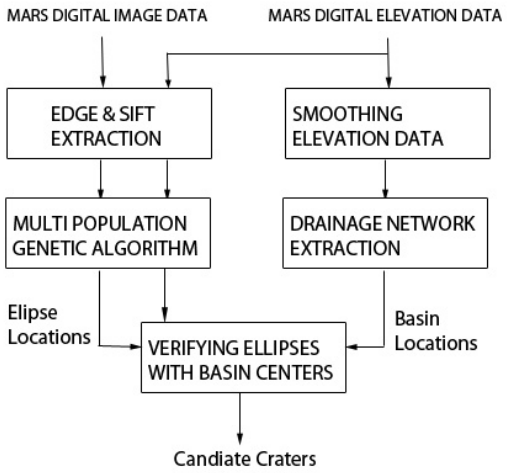



Figure 1: Overview of crater detector system

# 3 CRATER EXTRACTION FROM DIM

As described in the overview section of this document, we employ scale invariant feature transform and multi population genetic algorithm to find impact craters from optical data. In this section, we will give a detailed description of both algorithms involved.

## 3.1 Scale Invariant Feature Transform

Although the existing body of research on impact crater detection focus on extracted edges, we have also implemented SIFT algorithm which aims to reliably identify scale invariant features of an object proposed by Lowe [Low99]. Compared to the edges extracted, SIFT features are well localized around the rims of the craters as seen in Figure (2). The method that Lowe has proposed transforms image into collection of feature vectors that are invariant to scaling, rotation, and illumination changes. SIFT algorithm involves four main stages, which are scale-space extrema detection, keypoint localization, orientation assignment, and keypoint descriptor. Keypoints are defined as the extrema points of Difference of Gaussians (DoG) that occur at multiple scales. The reason of using DoG instead of gaussians is to gain efficiency. The algorithm eliminates outliers by discarding low-contrast keypoints and edge responses.

SIFT features have gained popularity in computer vision domain due to its successful applications in feature matching. Recently, SURF (Speeded Up Robust Features) , a faster version of the SIFT algorithm has been proposed which is based on Haar Wavelet responses [Bay08]. Although SURF feature detector is faster than SIFT, a comparative study between SURF and SIFT reveals that SURF features are not stable against rotation and illumination changes [Jua09]. This is the main reason we have used SIFT features in our study of a crater detector. High rotation variation between craters and illumination changes are possible due to the Mars surface, image acquisition equipment used, and the camera parameters involved.

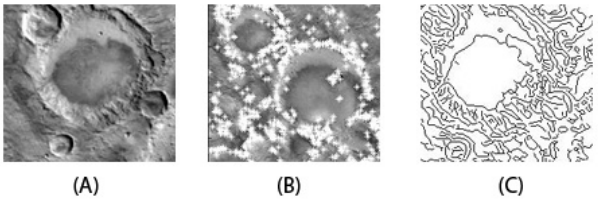



Figure 2: (A) Optical data obtained from Mars surface (B) SIFT features highlighted (C) Edges extracted by canny edge detector

## 3.2 Genetic Algorithm Variants for Ellipse Detection

In our implementation of crater extractor from DIM, craters are assumed to have elliptic shape. Since SIFT features extracted from previous stage of our system are assumed to be scale invariant, elliptic assumption of the feature vector is reasonable.

Most methods to detect ellipses from images can be categorized into two major groups. These are Hough Transform (HT) based methods, and stochastic algorithms.

HT based methods perform a mapping from image space to parameter space. The optima's of parameters corresponds to instances of primitives. Although HT is highly accurate and feasible to use for primitives with small number of parameters, computational demands of the method grows exponentially along with the parameter number [Yin99]. Since we need to detect ellipses which have five arbitrary parameters, HT based methods are infeasible to use because of large parameter space involved.

Stochastic algorithms have also been applied for geometric primitive extraction on 2D images since primitive extraction has been shown to be an optimization problem [Rot93]. Most popular stochastic algorithm used for primitive extraction is genetic algorithm (GA). Inspired by evolutionary biology genetic algorithm tries to find an approximate solution to optimization problems. Instead of exhaustively searching parameter space as in the case of HT, GA iteratively refines population to cluster solutions around the global optima. Moreover, inherently parallel nature of GA can be exploited on parallel computing architectures to produce scalable algorithms. A number of researchers have already used this idea to cope with growing datasets [Deg10].

Although GA based techniques have inherent strengths over HT based methods, finding multiple instances of a geometric primitive can't be directly mapped into problem space of GA because it approximates a global maximum. However, in our crater detection implementation, we want to detect several locally maximum ellipses rather than finding the globally optimal ellipse in the image. This is the reason we have implemented a multi population genetic algorithm that is able to find several locally optimal ellipses in the given image.

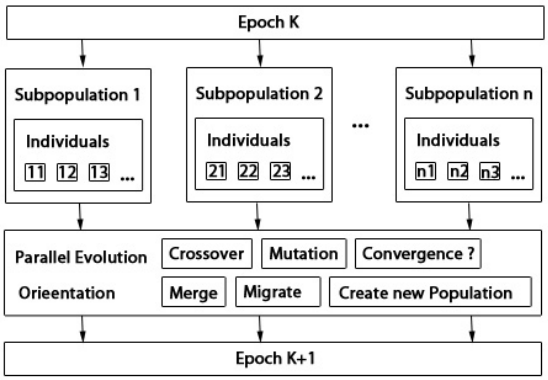



Figure 3: One iteration of Multi-Population Genetic Algorithm

The classical genetic algorithm implementation may also suffer from premature convergence. The term is used for harmfully fast convergence of a population to a suboptimal solution. The two commonly used solutions to this problem are fitness sharing and replacement of similar individuals. Both of the proposed modifications to genetic algorithm maintains the diversification of the population in order not to converge directly to a premature solution. The former, also called Sharing Genetic Algorithm (SGA), is proposed by Lutton et al. [Lut94]. SGA shares the fitness of similar individuals to decrease clustering around a single solution. The later, on the other hand, simply replaces the similar individuals with randomly generated ones to increase the diversity of the population. Although the replacement is necessary as the fittest individuals dominate the population, replacing with random individuals degrades the performance of the genetic algorithm since it may lead the population to an already searched space. Thus, SGA has inherent strengths over replacement of similar individuals method. Note that, SGA can also be used for the local optima search problem since it reduces the fitness values of individuals clustered around single optima.

The multi-population genetic algorithm (MPGA) is another variant of GA that can be used for multiple local optima detection. A number of subpopulations are generated and evolved in order to find several optima's. These subpopulations can be thought as islands where individuals can travel in between and create their own one. This adaptive clustering mechanism both concentrates the solutions around optimal points and diversifies the population across the search space. A research study conducted by Yao et al. investigates the use of both MPGA and SGA over the ellipse detection problem [Yao05]. Results of their study reveals that MPGA outperforms SGA in terms of both accuracy and performance. Following section of this document, describes the multi-population genetic algorithm used for ellipse detection on SIFT keypoints.

## 3.3 MPGA for Ellipse Detection

In the ellipse detection context, multi-population genetic algorithm evolves several populations aimed to represent ellipses from keypoints extracted. Figure (3) shows one iteration of MPGA where a number of populations evolve in parallel. Communication between subpopulations are performed through migration of individuals from one subpopulation to another. Creation of a new population is also possible when an individual does not exhibit an affinity with any existing populations. As the number of epochs increase, subpopulations can possibly replicate each other which would decrease the performance. To prevent this danger, our MPGA algorithm considers the merging the similar subpopulations.

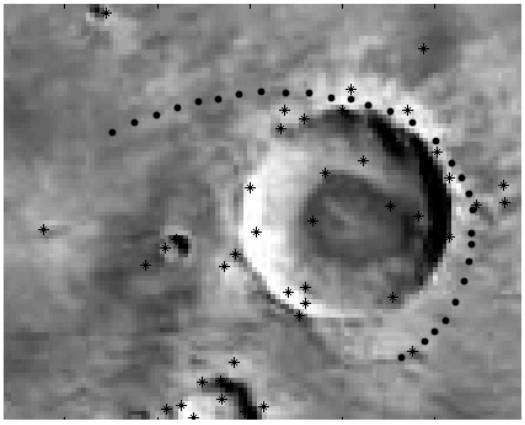


Figure 4: A chromosome that defines an ellipse over five keypoints

In the convergence case of a subpopulation, the keypoints of detected ellipse is removed from the image, and the individuals are deported. Note that, as the number of subpopulations decrease the number of individuals per population will increase. Thus, search will accelerate as the number of ellipses in the image decreases.

As seen in the Figure (3), MPGA can be characterized by a set of operations on individuals; crossover, mutation, fitness evaluation, and orientation. Orientation of an individual requires a set of operations for habitat selection, which are merging, migration, and new subpopulation generation. This section describes all stages of MPGA in the context of ellipse detection.

**Individual Representation** Individuals, also referred as chromosomes, are candidate ellipse parameters. As seen in Eq. (1), an ellipse can be represented with it's five arbitrary parameters.

$$p_0x^2 + 2p_1xy + p_2y^2 + 2p_3x + 2p_4y + 1 = 0 \qquad (1)$$

where (x,y) denotes the x and y coordinates of the feature points, and $p_{0..4}$ are parameters of the ellipse.

Given any five points $(x_i, y_i)$ where $i \in Z : 0 \leq i \leq 4$, parameters of an ellipse passing through them can be computed by solving five linear equations given in Eq. (2).

$$\begin{bmatrix} x_0^2 & 2x_0y_0 & y_0^2 & 2x_0 & 2y_0 \\ x_1^2 & 2x_1y_1 & y_1^2 & 2x_1 & 2y_1 \\ x_2^2 & 2x_2y_2 & y_2^2 & 2x_2 & 2y_2 \\ x_3^2 & 2x_3y_3 & y_3^2 & 2x_3 & 2y_3 \\ x_4^2 & 2x_4y_4 & y_4^2 & 2x_4 & 2y_4 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \vec{0} \quad (2)$$

Using this fact, the chromosome of an individual can be composed of five keypoints. In the literature, there are other individual representations for ellipse detection. Mainzer represents an individual directly by five parameters of the ellipse [Man02]. That is the parameters $p_{0..4}$ are encoded in the chromosomes of population. However, as Yao et al. noted that this representation generates a larger search space since the solutions may not even represent an existing ellipse. In our implementation, on the other hand, search is focussed on existing ellipses since chromosomes encode real keypoints extracted from optical images of Mars. In our implementation minimal point representation have been used for chromosome encoding. Figure (4) depicts an individual chromosome that is represented by a dashed ellipse over five keypoints extracted. Blue stars on the image shows the keypoints extracted by SIFT algorithm.

**Fitness Evaluation** In order to evaluate how fit the individual is, genetic algorithm requires a fitness function that returns a comparable value given a chromosome. Ellipse detection algorithms that involve the GA have widely match template around the ellipse represented by an individual. Mainzer et al. suggests fitness function at Eq. (3) that punishes edge pixels far from the ellipse [Man02] for each pixel (x,y) on the candidate ellipse.

$$f_1 = \sum_{x,y} \max_{\forall i,j} [E(x+i,y+j) - \frac{1}{c}(|i|+|j|)] \quad (3)$$

where $E(x,y) = \begin{cases} 1 & \text{if } Image(x,y) \text{ is an edge pixel} \\ 0 & \text{Otherwise} \end{cases}$

Considering that this operation has to be performed whenever a fitness of an individual has to be calculated, efficiency should be optimized. Distance map data structure stores the closest distance to an edge for each pixel in the original image. An approximation to a distance map can be realized by a set of morphological operations.

The research studies that aim to extract ellipses from 2D images rely on detected edges [Yao05]. However, imagery data obtained from the surface of the Mars exhibits high illumination variances and outlier edges. Instead of using only edge responses for evaluating the fitness of an individual, the keypoints extracted by SIFT algorithm have also been utilized. The distance map for SIFT features and the edges have been computed using morphological dilation with structuring image as 4x4 normal distribution. The fitness is then calculated as given in Eq. (4) that matches an ellipse around the set of feature points.

$$f_2 = w_1 f_1 + w_2 \sum_{x,y} \max_{\forall i,j} [S(x+i,y+j) - \frac{1}{c}(|i|+|j|)]) \quad (4)$$

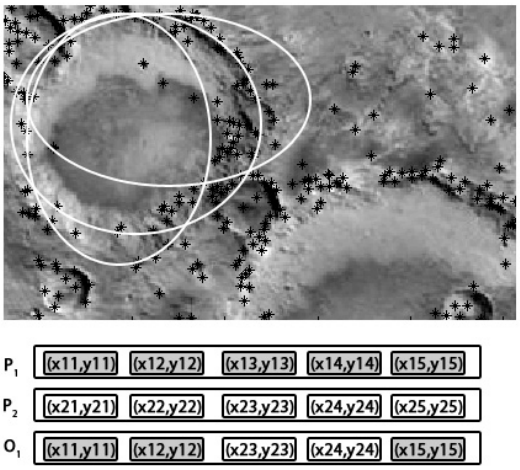



Figure 5: Uniform crossover operation over two individuals $P_1$ and $P_2$ which produces the offspring $O_1$

where $E(x,y)$ and $f_1$ are as in Eq. (3), $w_1$ and $w_2$ are weights determining the importance of edge response and SIFT features respectively. The equation for $S(x,y)$ is given in Eq. (5).

$$S(x,y) = \begin{cases} 1 & \text{if } Image(x,y) \text{ a SIFT keypoint} \\ 0 & \text{Otherwise} \end{cases} \quad (5)$$

**Merging of Subpopulations** As subpopulations evolve, converging ones may replicate in the population. In this case, all subpopulations evolve through a one globally optimal crater. To prevent replication, close subpopulations should be merged. Euclidean distance between cluster means can be used as closeness measure of two subpopulations. In literature, researchers have applied an empirical threshold over cluster distances to determine whether a merging operation should occur or not [Yao05]. However, scale variant distance measurements are not stable for small number of subpopulations. In our implementation, Mahalanobis distance is measured to check the merging condition.

While merge operations are begin performed, half of the fittest individuals are selected for a new subpopulation as suggested by Yao et al. [Yao05]. However, this implementation of merging operation causes the population to decrease. If a population undergo many merging operations, premature convergence problem may arise since the size would not be adequate to find all optima's. To prevent this side effect, size of each subpopulation is increased to compensate for the loss. The chromosomes of the introduced individuals are randomly generated from the set of keypoints.

**Migration & Splitting** On each evolution iteration of the population, chromosomes select the subpopulation with the least Mahalanobis distance. If a chromosome is not sufficiently close to any subpopulation, it creates a new subpopulation center of which is itself.

**Crossover** Uniform crossover has been implemented to produce offsprings. Since the individuals are represented by five feature points, the uniform crossover operation merely swaps the points of one parent with the other to produce an offspring. The offspring bares the subset of parents keypoints. Figure (5) shows the uniform crossover operation over two chromosomes.

**Mutation** The mutation operation is defined as randomly changing a gene of the chromosome and reassigning it to a new value. The operation is required to lead search towards uninhabitant areas. However changing one keypoint randomly generally results in degraded ellipse if the individual to be mutated is sufficiently fit. Therefore, mutation operation has to be enhanced to change more than one keypoints of the chromosome. In our implementation, a random number of keypoints have been replaced by mutated ones.

## 4 BASIN EXTRACTION FROM DEM

When two celestial bodies collide, a basin is usually formed at the larger colliding body. The abrupt height variation on the surface of planets survive longer than the rims of the basins which are degraded due to erosional processes. The optical data obtained does not carry any information about the height of the surface. Therefore, elevation data obtained from Mars surface have been utilized to find the basin locations. Researchers have proposed several approaches to find sink sources in elevation data. Most of the algorithms developed can be classified as either hydrological or morphological approach. The former approach uses the flooding algorithm of a water to detect sink sources, while the later recognize basins by their shape.

Since the impact craters on the Mars surface form topographic basins, hydrological algorithms outperform on basin location extraction. The survey of sink point extractors shows that the algorithm proposed by Callagnan et al. is being used commonly [Kis04]. In Callagnan's algorithm a rain drop is assumed in each cell of the elevation model with eight possible flowing directions [Cal84]. Due to predetermined flow direction for each cell, the algorithm is also called "Deterministic 8" (D8). The cell that the rain drop will flow into is determined by the slope of the eight possible flow directions.

Although Callagnan's algorithm provide simple and realistic flow simulation, the method fails on planar surfaces where surface runoffs are prevalent. To increase the reliability of the method, Freeman proposed multiple flow direction model that can find divergent flow drainage points by favoring water flow to several adjacent cells of lower elevation [Fre92]. The amount of water distributed from a higher elevation rain drop, $d_i$,is given in Eq. (6).

$$d_i = \frac{\max 0, S_i^w}{\sum_{j=1}^{8} \max 0, S_j^w} \tag{6}$$

where $S_i$ is the slope of adjacent cells, and $w$ is a constant factor determining the divergence of the flow.

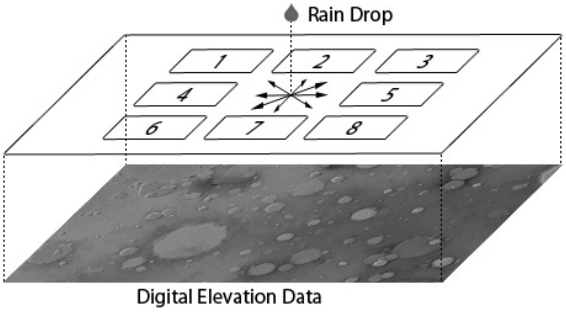



Figure 6: Sink source detection on Mars Digital Elevation Model

The distribution of the water drop is proportional to the slope of adjacent cells as Eq. (6) suggests. We have applied the multiple flow direction model to calculate the locations of the sink sources. Since the elevation data of the Mars surface is highly vibratile, the DEM data is first smoothed before the drainage networks are extracted. Figure (6) shows a rain drop with the flow directions on the Mars digital elevation data. Once the flow of water is stabilized, the resulting image of rain drop catchments is propagated to the last stage of our algorithm where the results of DEM and DIM data is fused.

## 5 MERGING RESULTS

The framework proposed have operated on two data set with two different algorithms. The result set is composed of the most fit ellipses extracted from DIM & DEM data and the image of basin locations obtained from DEM data. The figure (7) shows the set of ellipses extracted from optical and elevation data. The complementary nature of the results increases the robustness of the algorithm. Note that a portion of the ellipses extracted do not correspond to the craters. To decrease number of false positives, the basins extracted should be used to verify the ellipses.

To finalize the decision about the existence of impact craters, for each ellipse the ratio of the ellipse area and the catchment area under the ellipse is calculated. This metric is thresholded with fixed constant determined by our empirical studies. Finally the fittest ellipses extracted from DEM & DIM data are merged to compose candidate craters. To eliminate duplicate ellipses, the overlapping area is compared with the area of the bigger ellipse for each pair of ellipses. If the duplication is detected, the result of DEM data is output since ellipses obtained from DEM data have shown higher accuracies.

## 6 EXPERIMENTS & RESULTS

The test site we have selected for our experiments contains heavily cratered area that includes famous Herschel crater. The digital elevation and optical data is obtained from web map server (WMS) of NASA. Mars Digital Image Mosaic (MDIM) and Mars Orbital Laser Altimeter (MOLA) downloaded from WMS have the approximate bounding box as $7.42°, -18.42°, 172.02°, -7.58°$. The terrain chosen contains large number of degraded craters as well as non-crater

basins and other topographic structures. Another reason for choosing this area is the significant overlap over Barlow Catalog and the test sites previously chosen by researchers [Bue07].

The data retrieved from WMS is partitioned into 113 images of size 720x360 and overlapping ratio 1/4. The performance of the algorithm is measured by the metrics at Eq. (7-9). These metrics are proposed by Shufelt [Shu99] and have been used to measure the performance of crater detectors by a number of researchers [Bar04], [Kim05], [Bue07].

$$Detection = \frac{100TP}{TP+FN} \tag{7}$$

$$Branching = \frac{FP}{TP} \tag{8}$$

$$Quality = \frac{100TP}{TP+FP+FN} \tag{9}$$

In Eq. (7-9), TP, FP, and FN are abbreviations for True Positives, False Positives, and False Negatives respectively. The Detection metric measures the crater detection performance. The Branching metric measures the delineation performance. And Quality can be thought as measure of overall performance of the algorithm.

The researchers who have proposed crater detection algorithms have chosen different test sites. Some of them have even chosen test sites that do not include degraded craters [Kim05]. In order to test for reliability, we have selected a challenging terrain that includes highly degraded craters as in [Bue07]. The results are compared with both manually detected Barlow crater database and automatic crater detection algorithms proposed by researchers [Kim05], [Bue07], [Bar04]. The table 6 shows their findings. The D,B, and Q represents the metrics given in Eq. (7-9). Nontrivial test sites includes terrains where heavily degraded craters are common. The trivial test site used by Kim et al. includes only well-formed craters since they have noted that the algorithm is not capable of detecting the degraded craters [Kim05].

Our test site has more than 1/3 overlap with nontrivial test sites. The algorithm developed in this document had detected 621 craters in 113 segments. The number of non craters that were detected is 127. Thus, the Branching factor of our study is approximately 0.26. This is the lowest branching factor in the literature of impact crater detectors test on nontrivial test sites (see Table 6). Most false positives correspond to degraded rims of large impact craters with diameter $> 20$ km. The second best performing algorithm in terms of branching factor includes curvature profile calculation, basin detection, and hough transformation [Bue07]. Although Bue et al. have proposed a confirmation algorithm to verify the candidate craters found by Hough Transform, their verification strategy did not rely on separate set of calculations as in our case.

The algorithm we have proposed failed to detect 182 impact craters that are listed on Barlow Catalog. Most of the craters that our algorithm has failed to detect shows substantial deformations due to erosional processes. The detection rate is approximately 73% in our nontrivial test site. The rate of detection accomplished by this research is close to the best performing automatic crater detection algorithm in the literature [Bue07] and to the human detection rate [Bar88]. The overall quality metric of our algorithm is also approximately equal to the Bue's study with 61%.

| | D | B | Q | Test Site | Ref. |
|---|---|---|---|---|---|
| Bue | 74% | 0.29 | 61% | Nontrivial | [Bue07] |
| Barlow | 75% | 0.00 | 75% | Nontrivial | [Bar88] |
| Barata | 64% | 1.65 | 31% | Nontrivial | [Bar04] |
| Kim | 88% | 0.15 | 78% | Trivial | [Kim05] |

Table 1: Detection, Branching, and Quality metrics for different crater-detection algorithms

The comparisons in this section are made between similar test sites. The study of Kim et al. has not been compared to our study because of their simple test site selection. The metrics of the algorithm proposed in this document suggests higher quality when the test site is chosen not to include degraded craters.

## 7 CONCLUSION

This document describes an algorithm for Martian impact crater detection on Mars digital image and elevation data. Data fusion approach for the DEM and DIM is a contribution that improved the reliability of existing crater detectors. The use of Scale-Invariant Features and Multi-Population Genetic Algorithm is also novel for the literature of the crater detectors. The experimental results suggest a high detection rates close to the best performing algorithm and the most comprehensive crater catalog prepared manually. The improvements over the framework proposed are possible since the fitness evaluation procedure of MPGA can be complicated with other measures such as curvature profiles and heuristics. The adaptation of MPGA certainly introduces the flexibility that the current set of algorithms proposed lack. The complexity of fitness function can be traded with accuracy. It remains a future work for the authors to experiment with different fitness functions to optimize the performance of the algorithm. The aim of this study is to introduce a novel framework that is extensible and reliable to the literature of Hough Transform based algorithms.

## REFERENCES

[Bar88] N. G. Barlow, Crater size-distributions and a revised Martian relative chronology, Icarus, vol. 75, pp. 285-305, 1988.

[Kim05] J.R. Kim, J.P. Muller, S.V. Gasselt, J.G. Morley, G. Neukum, Automated Crater Detection, A New Tool for Mars Cartography and Chronology, Photogrammetric Engineering and Remote Sensing, vol. 71, No. 10, pp. 1205-1217, 2005.

[Ler01] B. Leroy, G. Medioni, E. Johnson, L. Matthies, Crater detection for autonomous landing on asteroids,

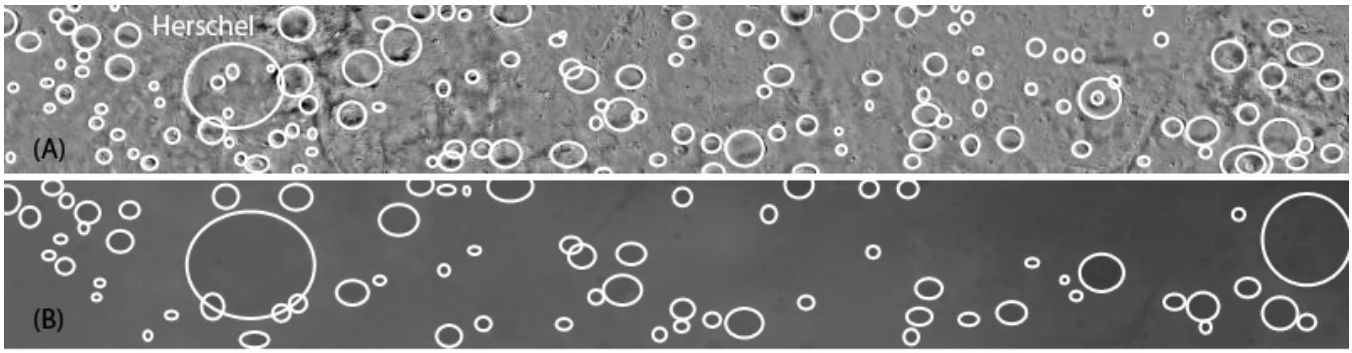



Figure 7: (A) The elllipses detected on Mars Digital Image Mosaic (B) The ellipses detected on Mars Orbital Laser Altimeter, both acquisited on 120.42° West, −18.42° South, 172.00° East, −10.58° North.

Image and Vision Computing, vol. 19, pp. 787-792, 2001.

[Wet05] Wetzler, P. G., Honda, R., Enke, B., Merline, W. J., Chapman, C. R., and Burl, M. C., Learning to Detect Small Impact Craters, in Proc. of the Seventh IEEE Workshops on Application of Computer Vision, vol. 01, 2005.

[Saw06] Y. Sawabe, T. Matsunaga, S. Rokugawa, Automated detection and classification of lunar craters using multiple approaches, Advances in Space Research, Volume 37, Issue 1, The Moon and Near-Earth Objects, pp. 21-27, 2006.

[Sug97] Sugiyama, T., Abe, K., 1997. Edge feature analysis by a vectorized feature extractor and in multiple edge. IEIC, D-2, J80-D-2, 6, pp. 1379-1389, 1997.

[Hon00] R. Honda, R. Azuma, Crater Extraction and Classifcation System for Lunar Images, technical report at Department of Mathematics, Kochi University, 2000.

[Bur04] M.C. Burl, P.G.Wetzler, Resource-constrained Application of Support Vector Machines to Sensor Data, Data Mining in Resource-Constrained Environments, 2004.

[Bue07] B. D. Bue, T. F. Stepinski Machine Detection of Martian Impact Craters From Digital Topography Data, IEEE Trans. Geosci. and Remote Sens., vol.45, pp.265-274, 2007.

[Mar04] B. L. Marchetti, L. Bruzzone, L. Lizzi, P. G. Marchetti, J. Earl, M. Milnes, Recognition And Detection Of Impact Craters From Eo Products, 2004.

[Low99] D. G. Lowe, Object Detection from local scale-invariant features, Proceedings of the International Conference on Computer Vision, pp. 1150-1157, 1999.

[Cal84] J. F. O'Callagnan, D. M. Mark, The extraction of drainage networks from digital elevation data, Comput. Vis. Graph. Image Process., vol. 28, no. 3, pp. 328-344, Dec. 1984.

[Bay08] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, SURF: Speeded Up Robust Features, Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346-359, 2008.

[Jua09] L. Juan, O. Gwun, A Comparison of SIFT, PCA-SIFT and SURF, International Journal of Image Processing (IJIP) Volume(3), Issue(4) pp 143-152, 2009.

[Yin99] P.Y. Yin, A new circle/ellipse detector using genetic algorithms, Pattern Recognition Letters, Volume 20, Pages 731-740, Issue 7, 1999.

[Rot93] G. Roth and M. D. Levine, Extracting geometric primitives, Comput. Vision. Graphics Image Processing: Image Understanding, vol. 58, pp. 1-22, 1993.

[Lut94] E. Lutton, P. Martinez, A genetic algorithm for the detection of 2D geometric primitives in images. Proceedings of the 12th international conference on pattern recognition, pp. 913, 1994.

[Yao05] J. Yao, N. Kharma, P. Grogono, A multipopulation genetic algorithm for robust and fast ellipse detection , Pattern Analysis & Applications, vol. 8 pp. 149-162, 2005.

[Deg10] M.Degirmenci, Complex geometric primitive extraction on graphics processing unit, journal of WSCG, Winter School of Computer Graphics, Volume 18, No.1-3, pp. 129-134, 2010.

[Man02] T. Mainzer, Genetic algorithm for shape detection, Technical report no. DCSE/TR-2002 06, University of West Bohemia, 2002.

[Fre92] T.G. Freeman, Calculating catchment area with divergent flow based on a regular grid, Computer and Geoscience, pp.413-422, 1991.

[Kis04] R. Kiss, Determination of drainage network in digital elevation models, utilities and limitations, Journal of Hungarian Geomathematics, Volume 2, pp. 16-29,2004.

[Shu99] J. A. Shufelt, Performance evaluation and analysis of monocular building extraction from aerial imagery, IEEE Trans. Pattern Anal. Mach. Intell., vol. 21, no. 4, pp. 311326, 1999.

[Bar04] T. Barata, E. Ivo Alves, J. Saraiva, and P. Pina, Automatic recognition of impact craters on the surface of mars, in Proc. ICIAR, Porto, Portugal, pp. 489496, 2004.

# Toward Objective Segmentation Evaluation

Štěpán Šrubař
Department of Computer Science, FEI, VŠB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava, Czech Republic
stepan.srubar@vsb.cz

## ABSTRACT

Two different segmentations of the same image can be evaluated in many ways. One resulting number hardly generalizes all differences in segmentations. Moreover, common methods can evaluate similar segmentations as quite different. Proposed evaluation divides dissimilarity into granularity and border difference. Granularity difference represents number of segments while border difference evaluates a rate of agreement in delineating of objects in the image. Such approach evaluates segmentations more precisely and keeps natural meaning of both resulting values.

## Keywords

Segmentation evaluation, probabilistic Rand index.

## 1. INTRODUCTION

Images are segmented for detection of objects and their separation. There is no best segmentation for an image. Still, many people will segment the same image similarly to each other, thus there exist some common rules for segmentation. Evaluation of segmentation can be divided into two main classes. The first takes image and its corresponding segmentation, the second takes only two different segmentation of the same image. We will be interested here in the second category only.

Segmentation evaluation methods are often based on the number of pixels (or probability of pixels) that were incorrectly classified or differs in these two segmentations. Some methods computes distance of mis-classified pixels or border pixels to the nearest correct place. Number of segments for evaluation was also proposed. We can define evaluation using some feature of the image or segments, namely size of segments or its eccentricity. Mentioned methods were closely described and compared in [Fer06a, Jia05a, Zha96a].

The newest inventions were made in the first class of evaluation methods. It was shown recently in [Unn07a] that probabilistic Rand index (PRI) outperforms some other methods from the same class. That is the reason why I take PRI for comparison with proposed method.

Three images (see fig. 1) and their segmentations was used for testing and comparison. Figures 2 and 3 show segmentations of two different images. PRI of segmentations in figure 3 is 0.54 and PRI of the first segmentations in figure 2 and 3 is 0.607. The higher number, the higher correspondence. Evidently PRI is not suitable for segmentations where the same object is segmented into different number of segments. Many other methods suffers the same problem.

Figure 1: Images 100075, 100098 and 103041 from Berkeley segmentation database [Mar01a].

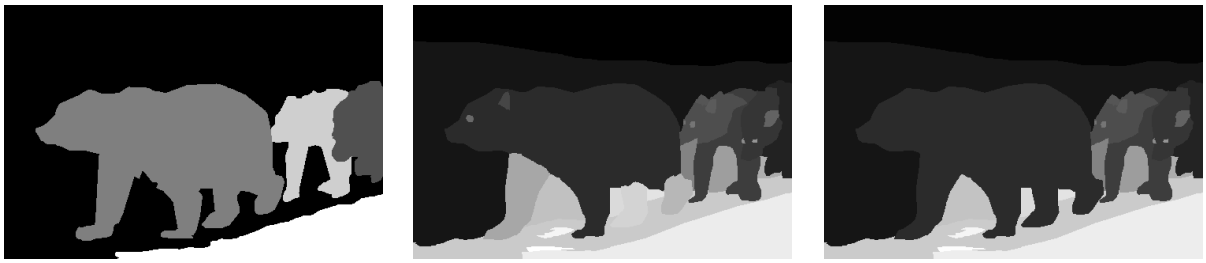


Figure 2: Two segmentations of the image 100075. Third segmentation is partial result of processing of second segmentation.

Figure 3: Two segmentations of the image 100098.

$$g(i) = -\log \frac{\sum_j |s_{ij}|^2}{\left(\sum_j |s_{ij}|\right)^2},$$

where $|s_{ij}|$ is the size of $j$-th segment of the image $i$. Logarithm converted unnatural geometric progression into more suitable arithmetic progression. This formula can be also used for arbitrary shaped part of an image, which will be used later.

## 2. SEGMENTATION DIFFERENCE

We cannot say, objectively, what the correct number of segments in segmentation should be. On the other hand the borders of the segments are created according to some rules which correspond to the borders of objects. Therefore, segmentation evaluation should measure the precision of borders and suppress the number of segments. Still, the number of the segments should be expressed by another number.

For simplicity, first segmenter can detect only main objects while the second segmenter can separate these objects into smaller ones (see figures 2 and 3). Still, the borders of main objects should correspond in both segmentations. First task is to group segments from the second segmentation to create equivalent representation of objects from the first segmentation. Such grouping expresses granularity difference. Having coarse second segmentation with the same number of segments as in the first segmentation, difference of borders can be then evaluated. Segmentation difference is then expressed as two numbers. Border difference corresponds to inaccuracy of borders of objects, while granularity difference expresses difference in resolution of objects.

### Granularity Difference

We can assume that bigger segments are better noticeable, thus they should have higher weight than smaller segments. Taking logarithm of normalized weighted sum of sizes of segments, we get following formula of granularity:

Result of *g(i)* is between zero and plus infinity. In numerator, the sum of sizes of segments is multiplied by their weights which are the sizes of the segments, thus the use of square. It is normalized by sum of weights (sizes of segments) and number of pixels of image. Both values are also identical which is represented by a square.

Say, we have one segment in the first segmentation representing some object. In the second segmentation, the same object is represented by more than one segment (see left bear in figure 2). We will call these segments in the second segmentation as joint segment. Such correspondence on an object in image will be called binding one to many segments or equivalently binding one segment to one joint segment. Another allowed bindings are one to one (trivial case of one to many), null to many and null to one (trivial of previous). For clearness see figure 4. Trivial cases will not be explicitly mentioned hereafter. Many to many binding is forbidden because it could lead to zero border difference for totaly different segmentations.

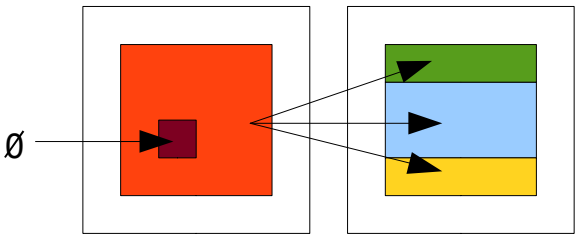



Figure 4: An example of binding in two images. Null to one is on the left, one to many binding is in the middle.

Pseudocode of searching of bindings for two images follows:

*I* ← find all intersections of segments

*B* ← null to many binding for each image

while I not empty

*i* ← remove the biggest intersection from *I*

if both segments from *i* unprocessed then

*B* ← create new binding from *i*

else if one segment from *i* unprocessed

if putting unprocessed segment into binding of processed segment will not create many to many binding then

put unprocessed segment into that binding

else

put unprocessed segment into corresponding null to many binding

mark segments from *i* as processed

Now we have all segments in one to many or null to many bindings. We can compute granularity difference and border difference. First, we need intersection in bindings to be able of computing granularity:

$$b_k = s_{1k} \cap s_{2k},$$

where $s_{1k}$ and $s_{2k}$ are joint segments that are bound. For null to many binding we make intersection of segments with whole image.

Suppose joint segment as an irregular shaped segmented image. Granularity of binding with that joint segment is defined as granularity of such segmented image. Resulting granularity difference is weighted sum of granularity of intersections:

$$gd(s_1, s_2) = \frac{\sum_k |b_k| \cdot g(b_k)}{\sum_k |b_k|},$$

where $|b_k|$ is number of pixels of binding $k$ and $g(b_k)$ is its granularity.

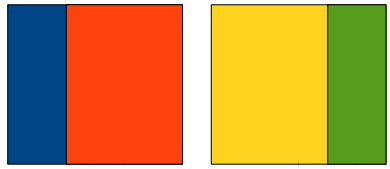


Figure 6: Segmentations for pseudonormalization of BD.

Time complexity, according to pseudocode and formulas, is $O(n)$ where $n$ is number of pixels of the image.

## Border Difference

Second value representing segmentation difference evaluates some type of precision of borders. For this purpose, we union segments in each joint segment to get one to one bindings only (see the union of a bear in the figure 2). Both null to many bindings are omitted in evaluation of border distance.

Each binding has now two corresponding segments. Border distance is based on sum of all pixels from one segment to the nearest pixels of the other segment. Proposed pseudonormalized border distance can be computed using following formula:

$$bd(s_1, s_2) = \frac{9 \cdot \sum_{b_i} [\sum_{x \in s_{1i}} d(x, s_{2i}) + \sum_{x \in s_{2i}} d(x, s_{1i})]}{w \cdot h \cdot max(w, h)},$$

where $b_i$ is binding, $s_{1i}$ is a segment from the segmentation $s_1$ and from the binding $b_i$, $d(x,s)$ is a distance of pixel $x$ to the nearest point of a segment $s$. I propose euclid distances due to radial symmetry. $w$ and $h$ represents width and height of the image respectively.

The outer sum is pseudonormalized. Special case, that was chosen for pseudonormalization, consists of two segments in both segmentations as seen in figure 6. One segment takes left third while the second segment takes the rest. The second segmentation is horizontally flipped case of the first segmentation. Border difference of such segmentation pair is $\frac{1}{9} w \cdot h^2$ (see the number 9 in previous fomula). Function *max* is used because similar case is created by rotation by 90° and we take the worse of these two cases. Such segmentation pair is not the worst case, thus we call it

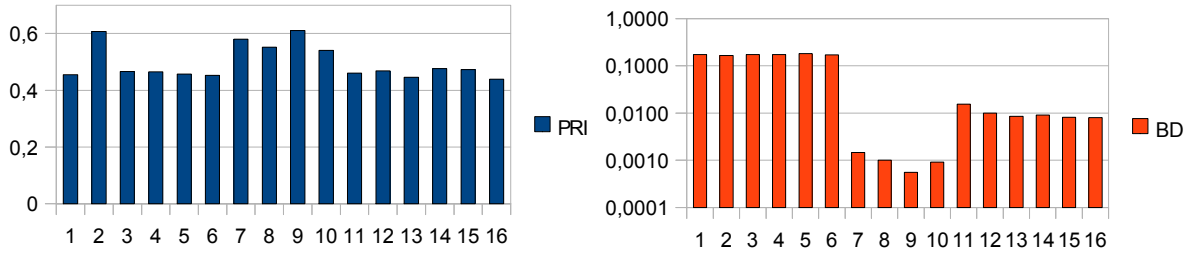



Figure 5: PRI and Border difference between 100098-1116 segmentation and 16 others. 1-6 are segmentations of image 100075, 7-10 are segmentations of image 100098 and 11-16 are segmentations of image 103041. PRI represents rate of correspondence, BD the rate of difference.

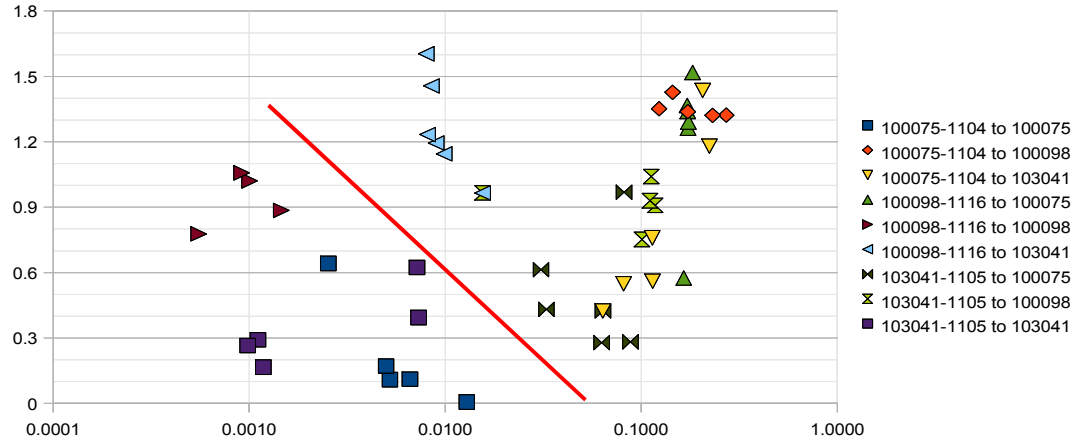



Figure 7: Similarity difference between three selected reference segmentations and segmentations of images 100075, 100098 and 103041. Horizontal axis represents border difference (in logarithmic scale) and vertical axis represents granularity difference. Red line separates points representing pairs of segmentations that belong to the same image. Points representing pairs from different images are on the right side of the red line.

pseudonormalization. On the other hand, typical values are not higher than pseudonormalization value.

Time complexity of border difference is $O(mn)$, where $n$ is number of pixels and $m$ is number of segments from both segmentations. Typically, the number of segments is much smaller than the number of pixels, thus the expected time complexity is $O(n)$. Running time of evaluation of granularity and border differences are in milliseconds on segmentations like in figures 2 and 3. Reference PRI method has time complexity $O(n^2)$ and runs tens of seconds on these segmentations to be computed precisely. For shorter computation time, PRI must use randomization algorithm Monte Carlo to estimate the result.

## 3. COMPARISON

I chose manually segmented images (see fig. 1) and all its segmentations. Proposed border difference (BD) is compared to probabilistic rand index (PRI) [Unn07a] in the figure 5. Images indexed as 7-10 should maximaly differ from others. Differences by BD are much greater than PRI and without any error (see index 2). Moreover, BD has to be represented in logarithmic scale.

Figure 7 shows robustness of proposed method. Similarity difference is measured for three chosen reference segmentations and the rest of the test set. All segmentations corresponding to their references are split from the other results by red line. The red line is diagonal, thus we need both granularity and border distance to make correct separation. Single value is evidently insufficient.

## 4. CONCLUSION

Two segmentations cannot be easily compared using a single number. Different number of segments does not necessarily mean that segmentations are from different images as the PRI could present. In fact, this can be caused by different granularity only. Thus, as was shown, granularity should be evaluated separately from precision of borders of segments. Such separation of properties in evaluation of segmentations leads to more robust results.

## 5. REFERENCES

[Fer06a] Fernando C. Monteiro and Aurlio C. Campilho. Performance Evaluation of Image Segmentation. ICIAR 2006, LNCS 4141, p. 248–259, 2006

[Jia05a] Xiaoyi Jiang, CyrilMarti, Christophe Irniger, and Horst Bunke. Distance Measures for Image Segmentation Evaluation. EURASIP Journal on Applied Signal Processing, vol. 2006, pp. 1–10, July 2005.

[Mar01a] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Proceedings of 8th IEEE International Conference on Computer Vision (ICCV '01), vol. 2, pp. 416–423, Vancouver, BC, Canada, July 2001.

[Unn07a] Ranjith Unnikrishnan, Caroline Pantofaru, and Martial Hebert. Toward Objective Evaluation of Image Segmentation Algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, June 2007.

[Zha96a] Y. J. Zhang. A survey on evaluation methods for image segmentation. Pattern Recognition, vol. 29, pp. 1335-1346, 1996.

# Knowledge representation using graph grammar rewriting system

Jiří Zuzaňák
Computer Graphics and Multimedia
Brno University of Technology
612 66, Brno, Czech Republic
izuzanak@fit.vutbr.cz

Pavel Zemčík
Computer Graphics and Multimedia
Brno University of Technology
612 66, Brno, Czech Republic
zemcik@fit.vutbr.cz

## ABSTRACT

Graph rewriting systems are applicable to vast majority of problems that are being solved in computer science. From problems concerning program optimization, software verification, description, and parsing of structured information to graph programming languages and layout algorithms. Graph rewriting systems are often represented as sets of rules describing transformations on graphs. The graph rewriting rule encapsulates complete information about applicable graph modification. In context of the described graph rewriting system, rules represent atomic modification of graph. A novel approach to graph rewriting and criteria for rule application enabling development of exhaustive graph rewrite system are introduced. The presented approach is derived from the well known double pushout approach (DPO). This paper concentrates on discussion of knowledge formalization representation for modeling concepts and on application of these concepts using the proposed programmed graph rewriting system.

## Keywords

Graph rewriting, Knowledge representation, Graph grammars, Image processing, Computer vision

## 1 INTRODUCTION

Most of the computing techniques can be simulated by graph rule based modifications performed on models. Systems from Chomsky hierarchy of grammars based on string transformations are used for modelling of various languages; similarly, graph rewrite systems (GRS) based on graph transformations can be used to transform models based on graphs.

This way, the proper set of graph transformations (graph rewriting rules) can be used to describe semantics of the simulated process. Such process can be e.g. program optimization, software verification, parsing of complex structured information, layout algorithms, or modification of complex networks. Traffic networks, simulation of chemical reactions, construction of artificial neural networks, etc. represents example of networks which can be modeled and transformed by graph rewriting systems. Follows few examples (described in more details) of application of graph rewriting systems.

In [11] traffic, networks are modelled by graph rewriting system. Result of graph rewriting process is Time Transition Petri Net (TTPN). Generated Petri Nets can be directly used for simulation of particular traffic situations. The graph grammars in this work define semantics of a given start model as all the reachable models that results from the application of rules.

Graph transformations are used also for simulation of chemical reactions. In [15] reactions are modelled by set of edge relabeling graph transformations. Reaction is based on transformation of substrate chemical graph to a product chemical graph. Transformation of is performed by breaking existing bonds and creating new bonds between molecule atoms. Atoms of chemical molecule are represented by vertices, while bonds between atoms are represented by graph edges.

Graph rewriting systems can be also exploited for rewriting of specialized patterns such are terms ([6, 2, 3]) and already mentioned strings. The theory of term graph rewriting studies the issue of representing finite terms as directed, acyclic graphs, and of modeling term rewriting by graph rewriting. Main advantage of this approach is sharing of common subterms explicitly, avoiding to copy subterm when applying rewrite rule with more than one occurrence of a variable in its right-hand side.

Graph rewriting is also used for performing basic computations (atomic steps) in graph programming (GP) languages [14]. GP languages are based on modification of input graphs, and related transformation of vertex and edge labels. Transformations are

determined by set of rewriting rules, where rules are part of graph algorithm. Resulted graph programming language is suitable for solving graph problems at a high level of abstraction, freeing programmers from handling low-level data structures.

Graph rewriting systems can be used as computational model for more general class of languages than just GP languages. Functional and logical programming languages for example. Whilst implementation of functional languages by graph rewriting is simple and intuitive, the implementation of logic programming languages is less direct and thus is more limited in practice. In [13] is proposed approach applying graph rewriting system as computational model for logic programming language. The achieved results are demonstrated on implementation of Prolog, and more novel logic programming language PLL.

In this paper, description of a novel approach for graph rewriting and transformation based on programmed graph rewriting system is presented. The graph rewriting system is designed mainly for parsing and interpreting of knowledge retrieved from image or video by various image processing and computer vision algorithms. However, presented graph rewriting system is designed in generally for use in any system which can benefit from graph transformations. The presented system and algorithms are designed especially for efficiency of execution.

This paper is organized as follows. In Section 2, basic definitions and notations are presented. Section 3 contains description of graph rewrite system basics (graph rewriting rules, left-side matching and rule application). Section 4 briefly describes proposed graph rewriting system. Implementation of described graph rewriting system is discussed in Section 5. Finally in Section 6 the discussion of the results and proposed ideas for future work concludes the paper.

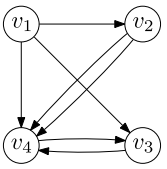
## 2 BASIC DEFINITIONS

In this section, definitions and notations further required for description of graph rewriting system will be introduced. Basic concepts of graph representation and graph properties and also relations among graphs will be described in details.

### Basic Definitions

For graph rewriting system and graphs itself to be usable, it is important to introduce a possibility to evaluate vertices and edges of graph by labels. Let labels be set of arbitrary objects of same class. Also, for further definitions, basic concepts of deterministic finite state machine will be needed.

**Definition 1 (Labels)** *Let $\mathcal{L}$ be a set of values, the* labels. *The relation equal $\subseteq \mathcal{L}\times\mathcal{L}$ determines equality of elements of $\mathcal{L}$. We will denote $(x,y) \in$ equal by $x =_{equal} y$*




| $\sigma_{t_G} \circ \sigma_{s_G}^{-1}$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $v_1$ | 0 | 1 | 1 | 1 |
| $v_2$ | 0 | 0 | 0 | 1 |
| $v_3$ | 0 | 0 | 0 | 1 |
| $v_4$ | 0 | 0 | 1 | 0 |

Figure 1: Example of directed multigraph

*Relation equal should be relation of equivalence (reflexive, symmetric and transitive).*

**Definition 2 (Deterministic finite state machine)** *The* deterministic finite state machine *(DFSM) is quintuple $M = (\Sigma, S, s_0, \delta, F)$, where $\Sigma$ is input alphabet, $S$ is non-empty set of states, $s_0 \in S$ is machine initial state, $\delta : S \times \Sigma \to S$ is deterministic state transition function, and $F \subseteq S$ is set of final states.*

### Graph Representation

Most important structure in graph rewriting systems is the graph itself. From now on, reference to a graph will refer to directed graph with multi-edges and loops enabled. (Directed multigraph with loops)

**Definition 3 (Directed multigraph)** *A* directed multigraph *$G$ is tuple $G = (V, E, \sigma_s, \sigma_t, \mu_v, \mu_e)$, where $V$ is set of vertices, $E$ is set of edges, $\sigma_s : E \to V$, and $\sigma_t : E \to V$ are functions mapping edges to theirs source and target vertices, and $\mu_v : V \to \mathcal{L}$, and $\mu_e : E \to \mathcal{L}$ are functions mapping vertices and edges to set of labels $\mathcal{L}$*

For text simplification we will write $G = (V_G, E_G, \ldots)$ for graph $G = (V_G, E_G, \sigma_{s_G}, \sigma_{t_G}, \mu_{v_G}, \mu_{e_G})$.

**Example 1** *Follows example of simple directed multigraph $G = (V_G, E_G, \sigma_{s_G}, \sigma_{t_G}, \mu_{v_G}, \mu_{e_G})$, where $V_G = \{v_1, \ldots, v_4\}$, $E_G = \{e_1, \ldots, e_7\}$, $\sigma_{s_G} = \{(e_1, v_1), (e_2, v_1), (e_3, v_1), (e_4, v_2), (e_5, v_2), (e_6, v_3), (e_7, v_4)\}$, and $\sigma_{t_G} = \{(e_1, v_2), (e_2, v_3), (e_3, v_4), (e_4, v_4), (e_5, v_4), (e_6, v_4), (e_7, v_3)\}$, and $\mu_{v_G} = \mu_{e_G} = \emptyset$. Example of graph is displayed in Fig. 1.*

**Denotation 1** *Several functions are implicitly associated with graph $G$*

*We say that edge $e$ is* incident *to $\sigma_s(e)$ and $\sigma_t(e)$, and if $\sigma_s(e) \neq \sigma_t(e)$ then vertices $\sigma_s(e)$ and $\sigma_t(e)$ are* adjacent.

*Set of all edges* incident *to vertex $v$ is defined by function $inc(v) = \{e \mid v = \sigma_s(e) \vee v = \sigma_t(e)\}$; similarly, set of all vertices* adjacent *to vertex $v$ is defined by function $adj(v) = \{u \mid (u = \sigma_s(e) \wedge v = \sigma_t(e)) \vee (u = \sigma_t(e) \wedge v = \sigma_s(e))\}$*

*The function $in : V \to \mathbb{N}$ determines the number of incoming incident edges of a vertex: $in(v) = |\{e \mid v = \sigma_t(e)\}|$. Similarly, the function out determines the number of outgoing incident edges: $out(v) = |\{e \mid v = \sigma_s(e)\}|$. A vertex with $in(v) = 0$ is called* root. *A vertex with $out(v) = 0$ is called* sink

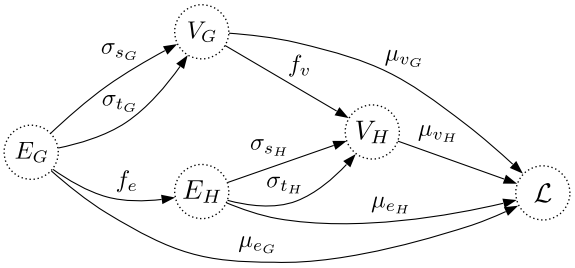



Figure 2: Graphs $G$ and $H$ and morphism $f : G \to H$

*The set of all graphs over set of labels $\mathcal{L}$ is denoted as $\mathcal{G}_\mathcal{L}$*

Several relations among graphs are defined. From basic graph unions and intersections to subgraph relation and graph isomorphisms, that are essential for graph rewriting systems.

**Definition 4 (Graph union and intersection)** *A graph $I = G \cup H$ is called* union *of graphs $G$ and $H$, if $I = (V_I, E_I, \ldots)$, where $V_I = V_G \cup V_H$, $E_I = E_G \cup E_H$, $\sigma_{s_I} = \sigma_{s_G} \cup \sigma_{s_H}$, $\sigma_{t_I} = \sigma_{t_G} \cup \sigma_{t_H}$, $\mu_{v_I} = \mu_{v_G} \cup \mu_{v_H}$, and $\mu_{e_I} = \mu_{e_G} \cup \mu_{e_H}$. Graph* intersection *$I = G \cap H$, and graph* difference *$I = G \setminus H$ are defined similarly*

**Definition 5 (Subgraph)** *A graph $H = (V_H, E_H, \ldots)$ is* subgraph *of graph $G$, if $E_H \subseteq E_G$, $V_H = \{v \mid e \in E_H \wedge (v = \sigma_{s_G}(e) \vee v = \sigma_{t_G}(e))\} \cup (V_{arb} \subseteq V_G)$, $\sigma_{s_H} = \sigma_{s_G} \cap (E_H \times V_H)$, $\sigma_{t_H} = \sigma_{t_G} \cap (E_H \times V_H)$, $\mu_{v_H} = \mu_{v_G} \cap (V_H \times \mathcal{L})$, and $\mu_{e_H} = \mu_{e_G} \cap (E_H \times \mathcal{L})$, where $V_{arb}$ is arbitrary subset of $V_G$. Subgraph is denoted by $H \subseteq G$*

**Definition 6 (Graph morphism and isomorphism)** *A* graph morphism *$f : G \to H$ between two graphs $G$ and $H$ consists of two functions $f_v : V_G \to V_H$ and $f_e : E_G \to E_H$ that preserve labels and attachment to vertices, that is, $\mu_{v_H} \circ f_v = \mu_{v_G}$, $\mu_{e_H} \circ f_e = \mu_{e_G}$, $\mu_{v_H} \circ \sigma_{s_H} \circ f_e = \mu_{v_G} \circ \sigma_{s_G}$, and $\mu_{v_H} \circ \sigma_{t_H} \circ f_e = \mu_{v_G} \circ \sigma_{t_G}$*

Functions illustrating connections of vertices and edges of graphs $G$ and $H$, and morphism between them $f : G \to H$ are depicted in Fig. 2.

The graph morphism $f$ is *injective* (*surjective*) if $f_v$ and $f_e$ are. If $f$ is both injective and surjective, then it is an *isomorphism*. In this case graphs $G$ and $H$ are isomorphic, which is denoted by $G \cong H$

**Definition 7 (Subgraph isomorphism)** *A* subgraph isomorphism *$f$ from graph $H$ to graph $G$ is graph isomorphism $f : H \to S$, where $S \subseteq G$. Denoted by $H \cong (S \subseteq G)$, or $f : H \to (S \subseteq G)$*

## 3 GRAPH REWRITING

Basic building block of graph rewriting system is graph rewriting rule, describing one possible modification (transformation) of the target host graph. In vast majority of literature ([11, 15, 14]) graph rule is represented by its left-hand side, right-hand side, and set of connections. Left-hand side of such rule can be represented by node, edge, or graph, from which rewriting system get its name: node-, edge-, graph-replacement systems. Right-hand side of rule is in most cases represented by graph. The connections describe relation between left-hand and right-hand side of rule. In many cases, connections also describe how to compute (find) labels of vertices and edges newly inserted into the host graph.

This paper is restricted to graph-rewriting systems, thus node and edge-rewriting systems are not considered. Most common approach for graph transformation is *Double Pushout Approach* (DPO) ([4, 7, 8]) which has rewriting rules of form:

$$r : L \leftarrow^{l} K \rightarrow^{r} R \tag{1}$$

where $L$ is left-hand side, $K$ is interface graph, $R$ is right-hand side, and $l$ and $r$ are morphisms. $K$ represents interface that is common for $L$ and $R$. Transformation of graph $G$ to graph $H$ by rule $r$ describe diagram in Equation 2.

$$\begin{array}{ccccc} L & \leftarrow^{l} & K & \rightarrow^{r} & R \\ \downarrow^{m} & & \downarrow^{d} & & \downarrow^{m^*} \\ G & \leftarrow^{l^*} & D & \rightarrow^{r^*} & H \end{array} \tag{2}$$

In order to apply rule to graph $G$ the match $m$ should be found between $L$ and $G$. In next step are from $G$ deleted all elements $L \setminus K$, thus producing graph $D$. Finally to produce graph $H$ elements from $R \setminus K$ are added to graph $D$.

**Definition 8 (Reducible expression)** *Reducible expression (redex) represents subgraph of host graph, to which is left side of rule mapped.*

Approach to graph rewriting presented in this paper is inspired by the DPO approach. But in contrary to DPO, the proposed approach works as follows: the first step of a rule rewrite, once a redex has been located, is to glue into the host graph new structure (represented by right-side); then change the shape of the graph by redirecting edges. Finally redundant structure (the garbage) is removed.

Beside the connections in existing implementations, also negative application conditions (NAC), are used, that are left-hand side context information restricting rule from application. This technique adds dependency on contextual information to process of left-hand side matching. Such extension of this process boosts expressive power of graph rewriting system.

In the presented approach left-hand side of rule is represented by general graph. This graph should be directed multigraph with loops, and must be weakly connected (in graph exist non oriented path from any vertex to each other vertex). Above mentioned constraint

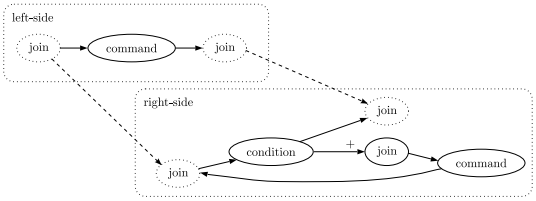



Figure 3: Example of simple graph rewriting rule

is requirement of algorithm for subgraph isomorphism search.

**Definition 9 (Graph rewrite rule)** *The quadruplet* $r = (L, R, E_{ex}, join)$ *is* graph rewrite rule*, which consist of* left-hand side $L \in \mathcal{G}_{\mathcal{L}}$*,* right-hand side $R \in \mathcal{G}_{\mathcal{L}}$*, set of* excluded edges $E_{ex} \subseteq E_L$*, and function* $join : V_L \to V_R$

The set of excluded edges $E_{ex}$ determines edges that must not be present in host graph in order to rule be applicable (NAC). Function *join* defines connection (interface) of graph rule in host graph. Example of simple graph rewriting rule is depicted on Fig. 3, where function *join* is expressed by dashed arcs, and $E_{ex} = \emptyset$.

**Definition 10 (rule matching)** *The graph rewriting rule* $r = (L, R, E_{ex}, join)$ *is applicable to host graph* $H$ *if:*

1. *graph isomorphism* $f_r : L_w \to (S_L \subseteq H)$ *exists, where* $L_w = (V_L, E_L \setminus E_{ex}, \sigma_{s_L}, \sigma_{t_L}, \mu_{v_L}, \mu_{e_L})$.
2. *the following holds for every* $u_l \in V_L$ *and* $v \in V_H$*:* $v = f_{r_v}(u_l) \wedge (in_{L_w}(u_l) < in_H(v) \vee out_{L_w}(u_l) < out_H(v)) \Rightarrow u_r \in V_R \wedge (u_l, u_r) \in join$
3. *the following holds for every* $e_{ex} \in E_{ex}$ *and* $e \in E_H$*:* $\sigma_{s_H}(e) \neq f_{r_v}(\sigma_{s_L}(e_{ex})) \vee \sigma_{t_H}(e) \neq f_{r_v}(\sigma_{t_L}(e_{ex})) \vee \mu_{e_H}(e) \neq_{equal} \mu_{e_L}(e_{ex})$

Graph $L_w$ is constructed by removing edges of set $E_{ex}$ from graph $L$. Graph isomorphism $f_r$ is called *occurrence*, and graph $S_L = (V_{S_L}, E_{S_L}, \ldots)$ is called *redex* (reducible expression). Statement 2 assures that all vertices of graph $L$ that are not in *join* relation with some vertex in graph $R$ have the same output and input degree as their image in graph $H$. Statement 3 assures that in graph $H$, edges that can be interpreted as images of edges from $E_{ex}$ are not present.

**Definition 11 (Graph rule match)** *The* Graph rule match *is a triplet* $m = (r, H, f_r)$*, where* $r$ *is matched graph rewriting rule,* $H$ *is host graph, and* $f_r$ *is matching morphism, which meet all conditions from Definition 10*

Evaluation function determines labels of new elements (vertices, edges) in host graph. This function is evaluated for every element $e \in V_R \cap E_R$ of right-side graph $R$. Every of these elements has associated one of evaluation heuristics.

**Definition 12 (Evaluation function)** *The method for determining labels of new elements of host graph is encapsulated in evaluation function* $eval : (e, m) \to \mathcal{L}$*, where* $e \in V_R \cup E_R$*, and* $m$ *is graph rule match*

### Rule application

Host graph $H$ is transformed into graph $I$ by rule $r$ in following steps:

1. Construct graph $I_1 = (V_{I_1}, E_H, \sigma_{s_H}, \sigma_{t_H}, \mu_{v_{I_1}}, \mu_{e_H})$, where
   - $V_{I_1} = V_H \cup V_R$
     - let $f_{n_v} : V_R \to V_{I_1}$ is injective function mapping right-hand side vertices to new vertices in host graph
   - $\mu_{v_{I_1}} = \mu_{v_H} \cup \{(v, l) \mid v_r \in V_R \wedge v = f_{n_v}(v_r) \wedge l = eval(v_r, m)\}$

   Vertices from right-side of rule are inserted to host graph. Labels of vertices are evaluated, and function mapping original right-side vertices to new vertices of host graph is constructed.
2. Construct graph $I_2 = (V_{I_1}, E_{I_2}, \sigma_{s_{I_2}}, \sigma_{t_{I_2}}, \mu_{v_{I_1}}, \mu_{e_{I_2}})$, where
   - $E_{I_2} = E_H \cup E_R$
     - let $f_{n_e} : E_R \to E_{I_2}$ is injective function mapping right-hand side edges to new edges in host graph
   - $\sigma_{x_{I_2}} = \sigma_{x_H} \cup \{(e, v) \mid e_r \in E_R \wedge e = f_{n_e}(e_r) \wedge v = f_{n_v}(\sigma_{x_R}(e))\}$
   - $\mu_{e_{I_2}} = \mu_{e_H} \cup \{(e, l) \mid e_r \in E_R \wedge e = f_{n_e}(e_r) \wedge l = eval(e_r, m)\}$

   for $x \in \{s, t\}$, (hence $\sigma_{x_G}$ stands for both $\sigma_{s_G}$ and $\sigma_{t_G}$). Edges of right-side of rule are inserted to host graph (edges are defined on already inserted vertices). Labels of edges are evaluated, and function mapping edges of right-side to new edges of host graph is constructed. Graph $I_2$ is depicted at Fig. 4.b.
3. Construct graph $I_3 = (V_{I_1}, E_{I_2}, \sigma_{s_{I_3}}, \sigma_{t_{I_3}}, \mu_{v_{I_1}}, \mu_{e_{I_2}})$, where
   - $\sigma_{x_{I_3}} = (\sigma_{x_{I_2}} \cup \{(e, f_{n_v}(v)) \mid (u, v) \in join \wedge e = \sigma^{-1}_{x_H}(f_{r_v}(u))\}) \setminus \{(e, f_{r_v}(u)) \mid (u, v) \in join \wedge e = \sigma^{-1}_{x_H}(f_{r_v}(u))\}$

   for $x \in \{s, t\}$. Edges connecting join vertices of left-side of graph rewrite rule are redirected to join vertices of right-side of graph rewrite rule. Resulting graph $I_3$ is depicted at Fig. 4.c., where redirected edges are displayed as dashed arcs.

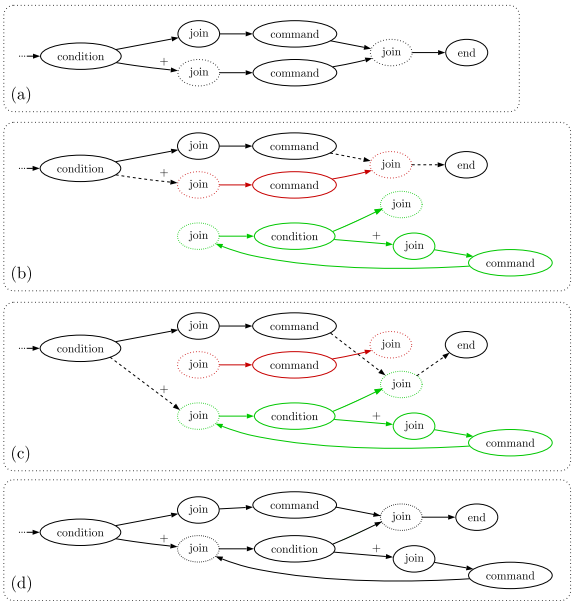



Figure 4: Illustration of host graph rewrite, driven by rule from Fig. 3

4. Construct graph $I_4 = (V_{I_1}, E_{I_4}, \sigma_{s_{I_4}}, \sigma_{t_{I_4}}, \mu_{v_{I_1}}, \mu_{e_{I_4}})$, where

   - $E_{I_4} = E_{I_2} \setminus \{e \mid e_l \in E_{L_w} \wedge e = f_{r_e}(e_l)\}$
   - $\sigma_{x_{I_4}} = \sigma_{x_{I_3}} \setminus \{(e,v) \mid e_l \in E_{L_w} \wedge e = f_{r_e}(e_l) \wedge v = \sigma_{x_H}(e)\}$
   - $\mu_{e_{I_4}} = \mu_{e_{I_2}} \setminus \{(e,l) \mid e_l \in E_{L_w} \wedge e = f_{r_e}(e_l) \wedge l = \mu_{e_H}(e)\}$

   for $x \in \{s,t\}$. Edges of rule left-side including their labels and mappings to source and target vertices are removed from host graph.

5. Construct graph $I = (V_I, E_I, \sigma_{s_I}, \sigma_{t_I}, \mu_{v_I}, \mu_{e_I})$, where

   - $V_I = V_{I_1} \setminus \{v \mid v_l \in V_L \wedge v = f_{r_v}(v_r)\}$
   - $\mu_{v_I} = \mu_{v_{I_4}} \setminus \{(v,l) \mid v_l \in V_L \wedge v = f_{r_v}(v_r) \wedge l = \mu_{v_H}(v)\}$
   - $E_I = E_{I_4}$, $\sigma_{s_I} = \sigma_{s_{I_4}}$, $\sigma_{t_I} = \sigma_{t_{I_4}}$, $\mu_{e_I} = \mu_{e_{I_4}}$

   Finally in last step vertices of rule left-side are removed from host graph. Graph $I$ (after removing left-side vertices and edges) is depicted at Fig. 4.d.

Description of application of rule is tightly connected to implementation. Vertices and edges of right-hand side are inserted to host graph before any left-hand side vertices or edges are removed, so labels of new vertices and edges can be determined. Rewrite of host graph by rule from Fig. 3 is inllustrated in Fig. 4.

## 4 GRAPH REWRITE SYSTEM

The literature concerning graph rewriting reports on various methods of organizing a collection of graph rewriting rules. These can be unordered, ordered, or event-driven. Choice of rule organization system largely affects the number of rewrite rule applications that must be tested during graph rewriting system execution. Parsing by graph grammar normally (without any rule organisation system) requires frequent testing of inapplicable rules. In contrast, an ordered graph rewriting system can directly transform an input graph into required output graph. Event-driven graph rewriting systems are highly time-efficient, applied rules are used only as direct response to external action.

### Unordered Graph-rewriting System

A set of graph rewriting rules. Rewrites the host graph by nondeterministically chosen rules until no further rule apply.

### Graph Grammar

A set of graph-rewrite productions. A starting host graph. A designation of labels as terminal or nonterminal. The starting graph is transformed by graph-rewrite productions until terminal graph is obtained. The set of terminal graphs that can be generated by this process is called language of the grammar (generative use). Parsing given graph: find sequence of rewrite productions that derive given graph from start graph (recognition use).

### Ordered graph rewriting system

A set of graph rewriting rules. A control specification (complete or partial ordering of rule-application). Rewrite the given host graph (choosing nondeterministically among applicable rules according to control specification) until a final state in control specification is reached.

### Event-driven Graph-rewriting System

A set of graph-rewrite rules. A externally-arising sequence of events. Rewrite the initial host graph: rewrite rules are executed in response to events.

Presented approach is restricted to first three of introduced categories of graph rewriting (respective graph grammar) systems. It is possible to extend presented rewriting system by event-driven execution of graph rewriting rules, but it was not introduced so far. Definitions of graph rewriting systems follows.

**Definition 13 (Graph rewrite system)** *The* graph rewrite system *(GRS), is represented by set of graph rewriting rules. In short it can be denoted just by $\mathcal{R}$.*

**Definition 14 (Graph grammar)** *The* graph grammar *(GG) is triple $\mathcal{X}_G = (\mathcal{P}, G_s, \mathcal{L}_t)$, where $\mathcal{P}$ is set of graph rewriting productions, $G_s \in \mathcal{G}_{\mathcal{L}}$ is starting host graph, and $\mathcal{L}_t \subseteq \mathcal{L}$ is set of terminal labels.*

Rules of graph rewrite system $\mathcal{X}$ are applied to host graph in nondeterministic order. This can result (for non confluent graph rewrite systems) in nondeterministic results. Based on this fact need arise to introduce ordering for rule applications of graph rewriting system.

**Definition 15 (GRS with priorities)** *The* GRS with priorities *is pair $\mathcal{X}_p = (\mathcal{R}, p)$, where $\mathcal{R}$ is set of graph rewriting rules, and $p : \mathcal{R} \to \mathbb{N}$ is function assigning priority number to each rule. This GRS represent simple version of ordered graph rewriting system.*

**Definition 16 (GRS driven by DFSM)** *The* GRS driven by DFSM *is defined as pair $\mathcal{X}_M = (\mathcal{R}, M_r)$, where $\mathcal{R}$ is set of graph rewriting rules, and $M_r = (\mathcal{R}, S, s_0, \delta, \emptyset)$ is deterministic finite state machine. A string accepted by $M_r$ defines string of the used graph rewriting rules. The set of applicable rules is in actual state $s$ given as $\mathcal{R}_s = \{r \mid (s, r) \to s_x \in \delta\}$. GRS driven by DFSM stops when $\mathcal{R}_s = \emptyset$, it works while any applicable rules exist. As in previous case this GRS presents ordered graph rewriting system.*

According to categorization of graph rewriting systems no explicit stopping conditions are introduced. The rules of graph rewrite system are applied while at least one matching of applicable graph rule exists.

# 5 GRAPH REWRITING IMPLEMENTATION

In this section the implementation of approach which was theoretically described in above text is described. Implementation of the proposed approach corresponds to the introduced theory.

## Graph Representation

The graph is represented by a dynamic structure that can be arbitrarily modified with no impact on performance. Graphs represented by proposed structure can be updated and modified using elementary steps consisting of edge and vertex removal and insertion.

In short, graph is represented by set of vertices and set of edges between these vertices. Each edge is described by two (source and target) indexes to the set of vertices and each vertex contains an array indexing all its incident edges (for faster computations). Each of graph elements (vertices, edges) has associated label which represents its evaluation. Described structure enables for simple execution of sophisticated graph algorithms, such as detection of spanning tree, testing bipartite graph, testing completeness, counting of graph components, search for shortest path, and other graph processing algorithms.

## Vertex and Edge Evaluation

The possibility to evaluate vertices and edges of graph by labels is provided by dynamically linked libraries defining operations over labels, such as comparison, copying, serialization, de-serialization etc. These libraries are created by graph rewrite system user, thus enabling use of arbitrary type of labels of vertices and edges of graph. For some conventional data types (integers, strings, etc.), default set of functions is defined.

## Graph Isomorphism Matching

From the definition of graph rewriting system, it can be seen that left-sides of graph rules are known a priory to their matching in host graph. This fact is exploited in system for detection of subgraph isomorphisms. Graph parsing automata is created representing subgraph isomorphism detector for left-side of each rule of graph rewriting system. Such automata is optimized for detection of common patterns in rules left-sides and exploits this knowledge in order to boost isomorphism detection speed.

The states of graph automata describes partial mapping of vertices and edges (spanning subtree) to a hypothetical host graph. The transitions between these states are described using edges, their properties, and by properties of target vertices (evaluation, in-out degree, directions, etc.). each of prototype subgraphs (left-sides of rules) has at least one final node, and by reaching of this node, the vertex mapping between this subgraph and host graph is decided.

Detection of subgraph and graph isomorphisms consists of two fundamental steps:

The first step is search for vertex permutations (mappings) of subgraph vertices to host graph vertices, thus describing set of functions $F_{vd}$. For which holds $F_v \subseteq F_{vd}$, where $F_v$ is set of functions describing proper subgraph vertex mappings. Detection of set of vertex mapping functions $F_{vd}$ is accomplished by search for a subgraph spanning tree.

The second step of subgraph (graph) isomorphism detection consist of search for edge permutations $F_e$. For each vertex mapping function $f_v \in F_{vd}$, possible permutations of edges are searched. Every found edge mapping function $f_e$ is added to set $F_e$; if $f_v \notin F_v$ then $f_v$ is inserted into set $F_v$, and the record $f_e \to f_v$ is inserted into the injective function $\pi$.

Result of this procedure is triple $(F_v, F_e, \pi)$, where $F_v$ is set of vertex mapping functions, $F_e$ is set of edge mapping functions, and $\pi : F_e \to F_v$ is function associating edge mapping function to vertex mapping function.

After detection of isomorphism of subgraph $L_w$ in host graph $H$, context conditions represented by set of excluded edges $E_{ex}$ are verified.

## Graph Rewriting System

Graph rewriting system itself is represented as set of graph rewriting rules. To set of graph rules are associated priorities or DFSM. Each GRS is described by one input text file. GRS file refers to rules represented in form of text file in graph format `.dot`, used by graph visualising library graphviz. Important part of graph rewriting system is represented by dynamic library, which defines vertices and edges evaluation functions.

# 6 CONCLUSION

In this paper, an approach for graph rewriting system implementation was proposed and also its real implementation was discussed. Graph rewriting system was designed for use in graph grammars, which are created for applications concerning parsing and describing of knowledge retrieved from image or video. In both forms of grammar use, either as generating tool, or for parsing of retrieved graph.

The advantage of proposed graph rewriting system is its ability to work with general directed multigraphs with loops. These graphs are represented by dynamic structure, which is not designed specifically for proposed task, but is designed more generally to enable to perform any tasks on them. No restrictions are introduced on vertex and edge labels, and their evaluation in process of rules applications. Proposed GRS was implemented in language C/C++ and its implementation respects its theoretical design.

Future work concerning graph rewriting system implementation will concentrate on further optimisation of graph rewriting algorithm and also optimisation of graph matching algorithm. The plan is to introduce categories of rules, determined by their analysis; insert, remove, and alter for example, and optimise rewrite technique (algorithm) separately for each of them. Parallel application of rules, based on analysis of redex overlays. In context of subgraph isomorphism search: isomorphism search based on incremental actualization of set of detected isomorphisms.

Further work will also concern examination of graph rewriting system properties. Graph rewriting system termination, Church-Rosser property, confluence, and from them resulting convergence should be discussed and evaluated.

A survey of possible applications of designed and implemented graph rewriting system, other than description or representation of image knowledge is also planned. Applications that can be possibly implemented and tested in context of graph rewriting system are for example (as was mentioned in Introduction): simulation of traffic networks, chemical reactions, automatic construction of artificial neural networks, etc.

# ACKNOWLEDGEMENTS

This work is supported by the European Commission under contract FP7-215453 - WeKnowIt.

This work was (also) supported by the project "Security Oriented Research in Information Technology" by Ministry of Eduction, Youth and Sports of Czech Republic no. MSM0021630528.

# REFERENCES

[1] A study of two graph rewriting formalisms: Interaction nets and MONSTR, February 20 1998.

[2] Zena M. Ariola and Jan Willem Klop. Equational term graph rewriting. *Fundam. Inf.*, 26(3-4):207–240, 1996.

[3] R. Banach. Transitive term graph rewriting, April 30 1996.

[4] R. Banach. The contractum in algebraic graph rewriting, April 30 1998.

[5] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Issues in the practical use of graph rewriting, December 18 1996.

[6] A. Corradini. A 2-categorical presentation of term graph rewriting, July 20 1997.

[7] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars (a survey). In *Graph-Grammars and Their Application to Computer Science and Biology*, pages 1–69, 1978.

[8] Hartmut Ehrig. Tutorial introduction to the algebraic approach of graph grammars. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 3–14, London, UK, 1987. Springer-Verlag.

[9] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. From graph grammars to high level replacement systems. In *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 269–291, London, UK, 1991. Springer-Verlag.

[10] P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *CoRR*, pages 176–187, 2001.

[11] Pieter J. Mosterman Juan de Lara, Hans Vangheluwe. Modelling and analysis of traffic networks based on graph transformation.

[12] Christoph Klauck. Graph grammar based object recognition for image retrieval. In *ACCV '95: Invited Session Papers from the Second Asian Conference on Computer Vision*, pages 561–569, London, UK, 1996. Springer-Verlag.

[13] Peter M and Peter M C Brien. Implementing logic programming languages by graph rewriting, April 22 1999.

[14] Detlef Plump. The graph programming language gp. In *CAI '09: Proceedings of the 3rd International Conference on Algebraic Informatics*, pages 99–122, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] Francesc Rosselló and Gabriel Valiente. Chemical graphs, chemical reaction graphs, and chemical graph transformation.

[16] Medha Shukla Sarkar, Dorothea Blostein, and James R. Cordy. GXL - A graph transformation language with scoping and graph parameters, September 12 1998.

[17] Vladimiro Sassone and Pawel Sobocinski. Coinductive reasoning for contextual graph-rewriting, February 02 2004.

[18] A. Schfürr. Programmed graph replacement systems. pages 479–546, 1997.

[19] Sjaak Smetsers. Term graph rewriting and strong sequentiality, November 06 1992.

# Graph Drawing in Lightweight Software: Conception and Implementation

Vitaly Zabiniako
Riga Technical University, Institute of Applied Computer Systems,
1/3 Meza street, Latvia,
LV-1048, Riga.
Vitalijs.Zabinako@rtu.lv

Pavel Rusakov
Riga Technical University, Institute of Applied Computer Systems,
1/3 Meza street, Latvia,
LV-1048, Riga.
Pavels.Rusakovs@cs.rtu.lv

## ABSTRACT

This work describes basic ideas of lightweight graph visualization system developed in Riga Technical University. Comparison of according existing freeware and shareware solutions is being made. Overall proposed software architecture at high abstraction level is presented along with details of implementation of its mechanisms. This work includes aspects of optimization of force-based graph layout algorithm; description of useful visualization techniques (such as projective shadows, visual data clustering that might be useful in design and analysis routines, etc.). Described ideas were implemented and verified by visualization of large graphs in original lightweight software 3DIIVE. Conclusions about achieved results are also presented.

## Keywords

Graph, visualization, architecture, algorithm, clustering.

## 1. INTRODUCTION

The concept of information visualization plays important role in modern IT industry, as it allows representing data according to the current needs of end-user and information processing tasks. There is a demand for these tools in such domains as data mining and analysis, education process, etc.

Multiple visualization solutions already exist, each with its own functionality and implementation specifics. Some of these tend to handle wide range of input data types and visualization tasks (which comes at the cost of complexity and usually – bulky architecture), while others are more of ad-hoc type solutions for specific purposes (and, as a result, non-usable outside originally intended visualization domain). The authors of this paper argue that there is a need for more agile solutions that should be based on achievements of modern computer graphics and object-oriented approach. These will provide appropriate aid for users in science, industry and business. Proposed architecture must ensure a set of primary features (i.e. ability to store and represent topology and associated metadata of a graph using appropriate description formats, layout algorithms and visualization techniques for better comprehension).

## 2. OVERVIEW OF EXISTING GRAPH DRAWING SOLUTIONS

Nowadays one can find wide range of tools that provide aid in graph drawing. Considering that it is impossible to summarize features of all these tools and versions, authors decided to analyze few distinctive representatives from both freeware and commercial products (summary of main aspects of such tools are presented in Table 1). Additionally, authors choose a set of criteria that is

| Criteria / Solution | Data format | Space | Graphics library | Platform | Additional tools |
|---|---|---|---|---|---|
| *Graphviz* | DOT (plain text-based) | 2D | Native | Windows / Linux / Mac OS | – |
| *Wilmascope 3D* | Native XML-based, GML | 3D | Java 3D | Windows | + |
| *aiSee* | GDL | 2D | Native | Windows / Linux / Mac OS | – |
| *Tulip* | Native, GML, DOT | 3D | Native | Windows / Linux | + |
| *yFiles* | Native, GraphML, GML | 2D | Native | Windows / Linux / Mac OS | + |
| *Walrus* | Native | 3D | Java3D | Windows / Linux / Mac OS | – |
| *Tom Sawyer* | Native | 2D | Native | Windows / Linux / Mac OS | + |
| *VGJ* | GML | 2D | Native | Windows / Linux | – |
| *3DIIVE* | Native XML-based | 3D | OpenGL | Windows | + |

**Table 1. Comparison of graph drawing solutions**

based on aforementioned primary features, support of additional analysis tools and implementation capabilities and allows comparing these solutions in order to get general vision of functionality.

## 3. OVERALL SOFTWARE ARCHITECTURE AND INDIVIDUAL FRAMEWORK COMPONENTS

Last row of Table 1 contains information about original lightweight graph visualization software system that is being developed in Riga Technical University for academic purposes as a part of doctoral thesis research. This system will be referred as "3DIIVE" (Three–Dimensional Interactive Information Visualization Environment) further in this text. This system was initially intended as utility program for demonstrating basic concepts in graph drawing area (focusing on drawing in three dimensions). As it can be perceived from the table, 3DIIVE is an agile solution that might be used both for general visualization of information encoded in form of graphs and more specific data-oriented applications in MS Windows platform. High abstraction level of proposed software framework is presented in Fig.1.

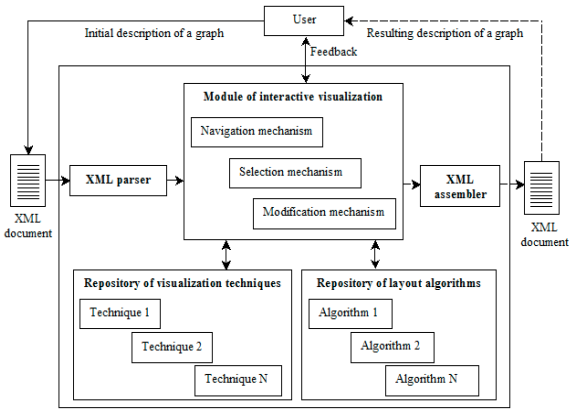



**Fig.1. Architecture of 3DIIVE system**

Our approach is based on the assumption that there must be clear separation in functionality of layout algorithms and visualization techniques – in this case these will be able to perform independently, making implementation of other algorithms / techniques much easier. Another assumption is that the structure of proposed framework must conform to the object-oriented approach, because graph itself can be conveniently interpreted as a set of topological and other associated properties that can be altered by appropriate methods.

Considering aforesaid, the main parts of this system are as follows: *XML parser* (interpretation of the input XML document with a description of the graph, and extraction of necessary information about topology of the graph and its elements), *module of interactive visualization* (the main part of the system that performs visualization by triggering necessary layout algorithms and visual techniques from repositories and relies on navigation, selection and modification mechanisms), *repository of layout algorithms* (a set of available layout algorithms for visualization of graphs), *repository of visualization techniques* (a set of techniques for improvement of comprehension) and *XML assembler* (acts similar to XML parser with only difference – opposite information processing flow).

The graph within a system is presented as a separate object with multiple attributes and methods. Information about topology of the graph from the parser module is converted into the adjacency matrix. The rest of the data is loaded into multiple arrays with references to associated elements.

## 4. IMPLEMENTATION OF LAYOUT ALGORITHMS AND VISUALIZATION TECHNIQUES

### 4.1 Layout algorithms

Current implementation relies on well-known force-based layout algorithm proposed by Peter Eades [Ead84]. The repository holds both its original and custom versions with optimizations done by authors. It is founded on combination of force-based and orthogonal layout properties and includes additional improvements.

For example, during the search of equilibrium state, in case if a value of average kinetic system energy achieves local minimum, it takes an additional time to retrieve from this, that's why authors make a step with random offset from current position, bypassing "slow" iterative recovering upon detection of local minimum. Another aspect is that the equilibrium state is being formed with unpredictable offset in space. In our system the model of graph is placed near the origin of a coordinate system.

Authors evaluated performance of both original ("FB") and modified ("MOD") algorithms with a set of time measurements from the first iteration and to the moment, when an equilibrium state was reached. Probability distribution is shown in Fig.2.

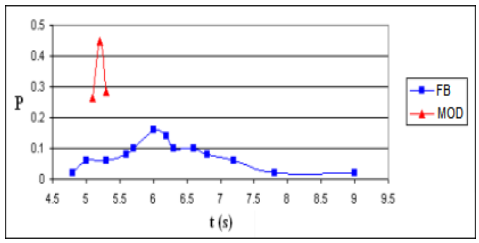



**Fig.2. Probability distribution graph**

Statistical processing allows concluding that the modified algorithm performs about to 15% faster while showing more stable distribution of time required for the execution. For detailed description of improved algorithm and according experiments refer to [Zab08].

## 4.2 Visualization techniques

Repository of visualization techniques holds a set of tools for improvement of information comprehension. It includes both common useful techniques, such as *transparency* (in case if certain part of the graph is chosen for further analysis, transparency helps to abstract from the other data by visually "weakening" it while still allowing to perceive the whole graph structure), *magnification* (interactive scaling up visuals that is implemented via image post-processing and allows to see more details by increasing resolution of these), etc. For more detailed description refer to [Zab09].

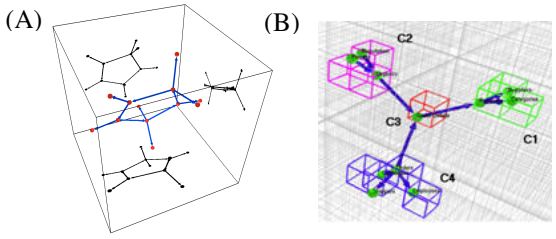



**Fig.3. "Projective shadows" and "Visual clustering"**

In 3DIIVE there are few custom visual techniques implemented by authors. One of such techniques is "*Projective shadows*" (Fig.3, part A). The idea is to draw the three-dimensional data model in iterative steps. The first step captures the model from the current position of the virtual camera like in previously mentioned techniques. During next steps camera is placed so that it faces model orthogonally – directly from the top, front, left etc. Each rendering result is placed into separate texture. When all steps are complete, textures are placed on corresponding faces of the rectangular parallelepiped (or cube). The parallelepiped is drawn in the scene so that three-dimensional data model is situated at its centre. In this case each texture represents a projection or essentially a "shadow" of original data structure.

As the result, user perceives not only the graph itself, but he can also evaluate and choose one two-dimensional instance which suits for outputting it in a plane surface (for example – for printing the graph on a paper). Two-dimensional instances are updated each frame and modifications that user performs with the spatial graph are reflected in all projections in real-time which makes this technique particularly useful.

In order to support this technique, graphical framework must provide access for rendering to texture which can be applied to the scene later. OpenGL framework (this API has been chosen due to its simple yet effective state machine model [Zab06]) allows to implement projective shadows with the code as follows:

```
/*1*/ ... //set matrix for camera
/*2*/ ... //draw object
/*3*/ for (x=1; x<=DimNum; x++)
/*4*/ {
/*5*/ ... //set matrix for dimension
/*6*/ ... //draw object
/*7*/ glCopyTexImage2D(GL_TEXTURE_2D,
          0,GL_RGBA,0,0,512,512,0);
/*8*/ }
/*9*/ //draw cube with textured faces
```

Projective shadowing allows to get more detailed comprehension about data structure – when perceiving spatial model from particular point of view it is sometimes hard enough to judge how complex the graph topology in individual dimensions is. Multiple projections allow to get rid of this problem. This concept is similar to orthographic projection views in CAD (Computer-Aided Design) systems. The difference is that in this case shadows allow to explore topological relationships of multiple elements rather than purely geometrical properties of single object. It is also suitable for tasks where the result of visualization must be presented or printed out as planar image.

Another custom implemented technique is in relation with graph data *clustering* task (Fig.3, part B). In general, clustering is required in case if data units must be evaluated in terms of its similarity or semantic closeness [Wri04]. Two common examples of such analysis are optimization of storing data in memory (in the field of database technology classic implementation of "Many-to-Many" relationship between two entities via auxiliary table results in data storage in different memory regions, so unwanted additional fetching of memory pages would be required) and deriving hidden patterns in large loosely structured data sets (detecting of dense semantic relationships among entities of knowledge domain may influence management strategy and even trigger development of new business rules).

Data clustering opportunity in 3DIIVE is based on secondary effect of force-based approach: the final layout tends to group densely interconnected nodes close to each other, while separating loosely connected groups in different space regions. A space partitioning mechanism is required to get the desired results (separate sets containing unique data

elements). The implemented model of space partitioning is based on octree that produces recursive division of cube model as shown in Fig.4.

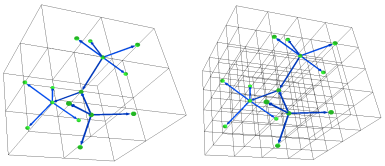


**Fig.4. Recursive space partitioning model**

Finally, the last implemented mechanism that is needed for accomplishment of clustering is regions merging – the natural way to bring multiple close nodes into corresponding cluster. Definition of a cluster with a set of neighbor regions is recursively transitive by its nature – region *R*, its neighbors *{R'}*, neighbors of neighbors *{R''}*, …, etc. belong to the same class. Authors propose the pseudo-code for the algorithm for assigning unique cluster identifications for an ordered set of regions:

```
proc assign_cluster_id
  for each region r process_region(r);
end

proc process_region(r)
  if r = Ø or r has id then return;
  if r has neighbors with id then
        assign same id to r;
  else
        assign new id to r;
  end
  for each neighbor n process_region(n);
end
```

The first procedure *assign_cluster_id* initiates sequential processing of regions by calling subroutine *process_region*. When non-empty and un-marked region has been found, it becomes either a core of new cluster or a part of already existing one – depending on the state of surrounding regions. Then all its neighbors are recursively revisited as in Fig.5 (sequence of regions is marked with red numbers and resulting clusters are marked with blue).

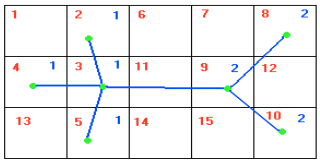



**Fig.5. Example of partitioning sequence**

## 5. USAGE EXAMPLES

Screenshots of different 3DIIVE functioning modes while visualizing complex network that consists of few hundred nodes are presented in Fig.6, (part A – selection step, B – usage of magnification, C – usage of transparency, D – clustering step).

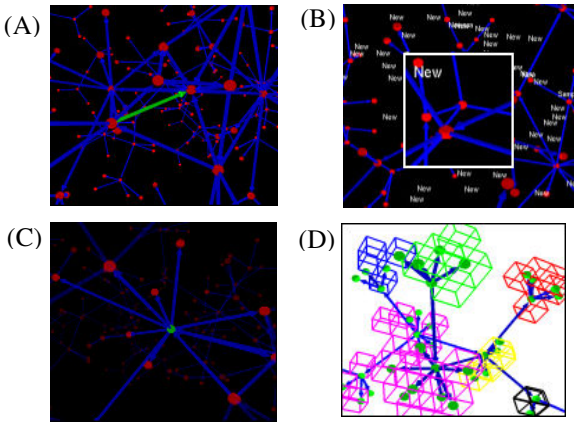



**Fig.6. Screenshots of 3DIIVE workflow**

## 6. CONCLUSION

Our implementation – 3DIIVE software system, allows visualization, analysis, editing of general graphs and provides a set of useful techniques for better information comprehension.

Current version holds both well-known existing graph drawing solutions and those proposed by the authors. The functional content of each module is gradually expanded during author's researches in the domain of graph visualization.

Considering its clustering capabilities this system can be used not only as a system for visualization and analysis but also as a tool for optimization, e.g. for refining relational model of database structure.

## 7. REFERENCES

[Ead84] P. Eades. A heuristic for graph drawing, Congresses Numerantium, vol. 42, pp. 149-160 (1984).

[Wri04] M. Kaufmann, D. Wagner "Drawing Graphs: Methods and Models", Springer, 312 p. (2001).

[Zab06] V. Zabiniako, P. Rusakov. Comparative Analysis of Visualization Aspects in Technologies Direct3D and OpenGL. Scientific Proceedings of Riga Technical University, Computer Science, series 5, vol. 26, pp 209-221 (2006).

[Zab08] V. Zabiniako, P. Rusakov. Development and Implementation of Partial Hybrid Algorithm for Graphs Visualization, Scientific Proceedings of Riga Technical University, Computer Science, ser. 5, vol. 34, pp. 192 – 203 (2008).

[Zab09] V. Zabiniako, P. Rusakov "Supporting Visual Techniques for Graphs Data Analysis in Three-Dimensional Space". The 50th Scientific Conference of Riga Technical University, Computer Science, Applied Computer Systems, October, Riga, Latvia (2009).

# Clifford Algebra and GIS Spatial Analysis Algorithms – the Case Study of Geographical Network and Voronoi Analysis

Zhaoyuan Yu
Key Laboratory of VGE
Nanjing Normal University
No.1 Wenyuan Road
210046,Nanjing,Jiangsu,China
yuzhaoyuan@163.com

Linwang Yuan
Key Laboratory of VGE
Nanjing Normal University
No.1 Wenyuan Road
210046,Nanjing,Jiangsu,China
yuanlinwang@njnu.edu.cn

Wen Luo
Key Laboratory of VGE
Nanjing Normal University
No.1 Wenyuan Road
210046,Nanjing,Jiangsu,China
luow1987@163.com

## ABSTRACT

Introducing Clifford Algebra as the mathematical foundation, we proposed a unified multi-dimensional GIS data model, constructed by linking data objects of different dimensions within the multivector structure of Clifford algebra. Then, algorithms for geographical network analysis (such as shortest path, minimum unicom and maximum flow analysis) and high-dimensional voronoi diagram were constructed. We use both simulation and real world data to test the usability and performance of our algorithms. The result gives very positive prospect of implement GIS analysis algorithms under Clifford Algebra framework. In that way, traditional GIS analyses algorithms can be extended not only accommodate various dimensions but also get beneficial on performance.

## Keywords

Clifford Algebra, GIS, Multi-Dimensional, Algorithm.

## 1. INTRODUCTION

Clifford algebra, which brings scalar and vector algebra into a unified framework and is widely used in various fields, brings us powerful tools to express multi-dimensional objects, extending classical geometric analysis and inherit and transplant existing analysis methods under unified framework. In this article, we introduce Clifford Algebra in GIS, and propose a unified multi-dimensional data model and several practical GIS analysis algorithms, which aim to evaluate the usability and efficacy of introducing Clifford algebra into multi-dimensional GIS spatial analysis. We also discussed their tests on both real world geographical and simulated data.

## 2. DATA MODEL

“Support data models for a complete range of geographic phenomena” and “support a wide range of types of geographic simulation” have already been identified as two of the 10 “grand challenges” for GIS [Lon05]. GIS data models should be developed for modeling complex geographic phenomena, whereas current GIS systems show limitations in representing real geographic phenomena [Yua09]. Some traditional GIS data models, separate objects of different dimensions and processing them separately, which increases the complexity of modeling and expressing real world data and splits and blurs the spatio-temporal semantics. Other models, like the event-based [Peu95] or the object-oriented data models [Wac94], also suffer dimensionality problems and compatibility with existing GIS data. Recently developed data models extend some of the above-mentioned aspects, but still meet a lot of difficulties in seamlessly integrating GIS data of different dimensions [Liu08].

We attempt to unify the multi-dimensional GIS data by associating the objects of different dimensions. The unified data model can be universally described as (x,y,z,a), where “a” means attributes that should be include in analysis process (Figure1). We can reorganize multidimensional GIS data by the combination of different dimensions, from two dimensions (x-a, y-a, z-a, etc.), three dimensions (x-y-a, x-z-a) to four dimensions (x-y-z-a). In order to analyze the characteristics, relationships and relative processes with Clifford algebra, we need to transform the geographical space into Clifford space. At last, based on the multi-dimensionality of the blades, we can construct corresponding data classes for the description of data objects with different dimensions with in a single multivector data structure, which can be further analyzed with various geometric operators.

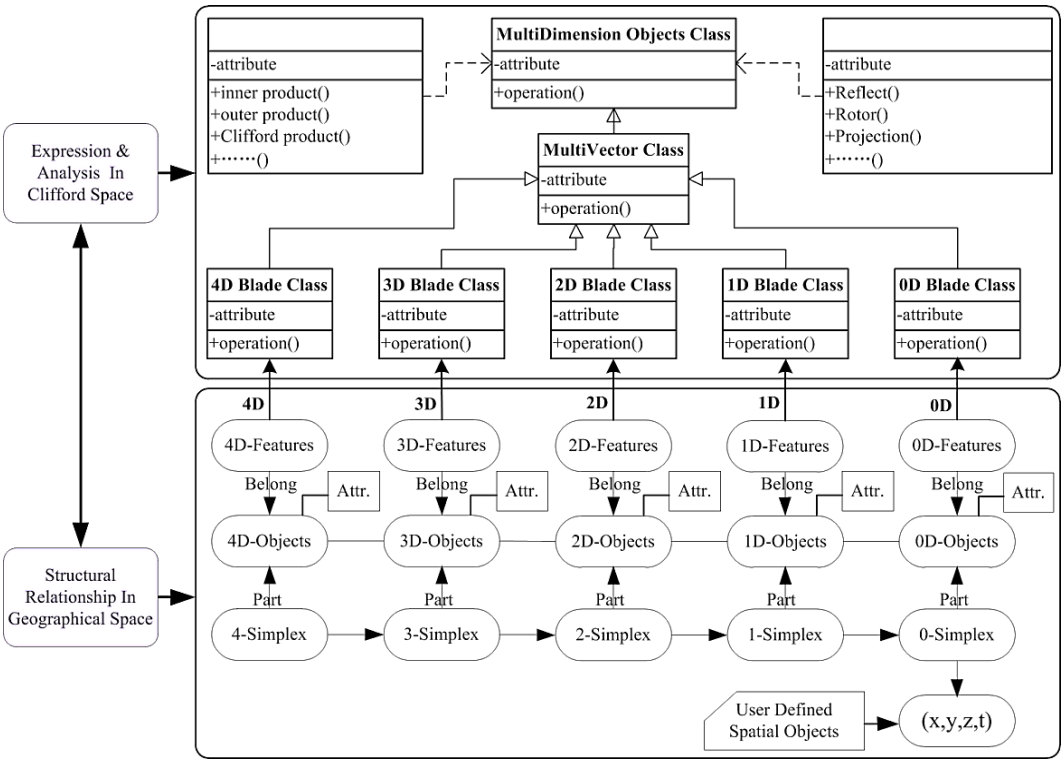



**Figure 1. The structure of unfiled spatial-temporal data model**

## 3. GEO-NETWORK ANALYSIS

Network analysis remains one of the most significant and persistent research and application areas in GIS. Many network analyses are among the most difficult issues to solve in terms of their combinatorial complexity [Cur07]. Some early researches on graph theory have proved that Clifford algebra can express the dynamic evolution of a directional and unidirectional finite graph, and can greatly reduce the computational complexity (e.g. [Sch10]). The geometric characteristics (direction, distance, topology and adjacent relationships) can be easily modeled by coding the nodes and routes with Clifford elements. We propose Geographical Network Analysis algorithms that can automatically construct the shortest path, minimum unicom and maximum flow from raw geographic networks without complex preprocessing. The flow chart of each algorithm was shown in Figure2. The above algorithms are implemented in C++ as a module of Unified Temporal-Spatial Analysis System based on Clifford algebra (CAUSTA) [Yua10].

We use China High-way data to test the correctness of our algorithms. The road network test result was shown in Figure3, comparing them with output of traditional GIS network analysis suggests the output of our algorithms and traditional GIS network algorithms are the same, which suggest our algorithms are correct. The spatial and temporal efficiency of our algorithms are computed using random network with different nodes and paths. Taken shortest path for example, the numbers of nodes varied from 1,000 to 10,000, and each network was repeated 10 times with random start and end nodes. Avoiding the use of the adjacent matrix simplifies the design and extension of the algorithm, so that our algorithm has much less memory usage, and can be extended for large scale network analysis (Table1). We also have two experiment with 50,000 and 100,000 nodes networks to test our algorithm capability of large scale data. Even with this large numbers of nodes, the 10 times means of time and memory usage are only 59.153s/851.172s and 12333Kb/27471Kb . 33Kb/27471Kb .

| N. Nodes | N. Paths | Dijkstra Time Mean(s) | Our Time Mean(s) | Our Time Stdev(s) |
|---|---|---|---|---|
| 1000 | 2866 | 0.049 | 0.043 | 0.003 |
| 3000 | 5890 | 0.533 | 0.123 | 0.005 |
| 5000 | 10428 | 1.506 | 0.345 | 0.023 |
| 8000 | 15414 | 3.964 | 0.671 | 0.045 |
| 10000 | 30244 | 6.652 | 2.298 | 0.117 |

**Table 1. Test Networks statistics and time performance compared with Dijkstra algorithm**

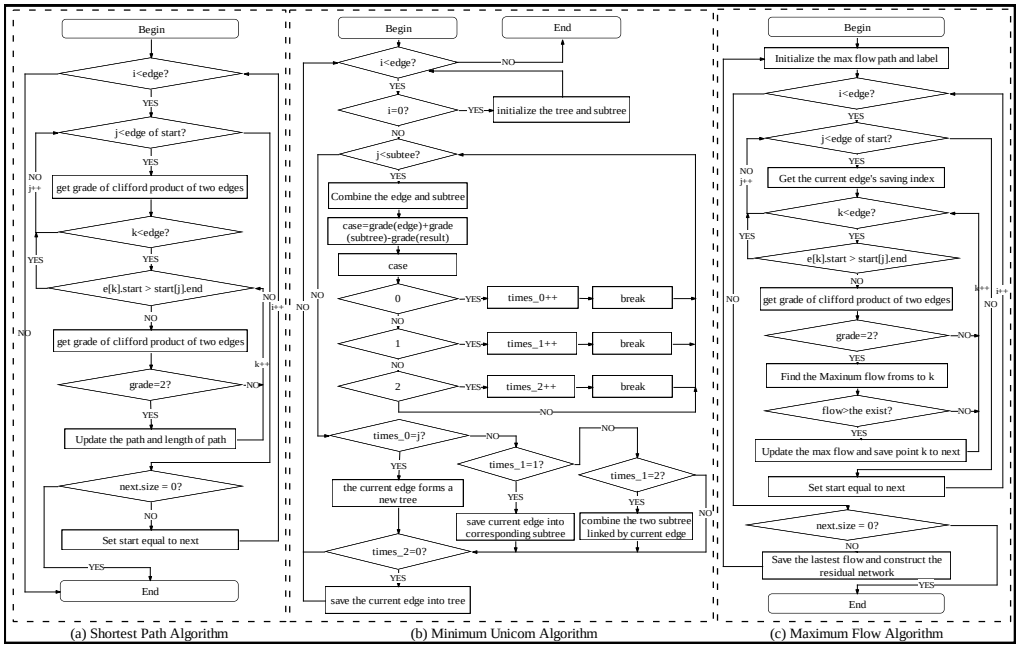



**Figure 2. Flow Chart of Network Analysis Algorithms**

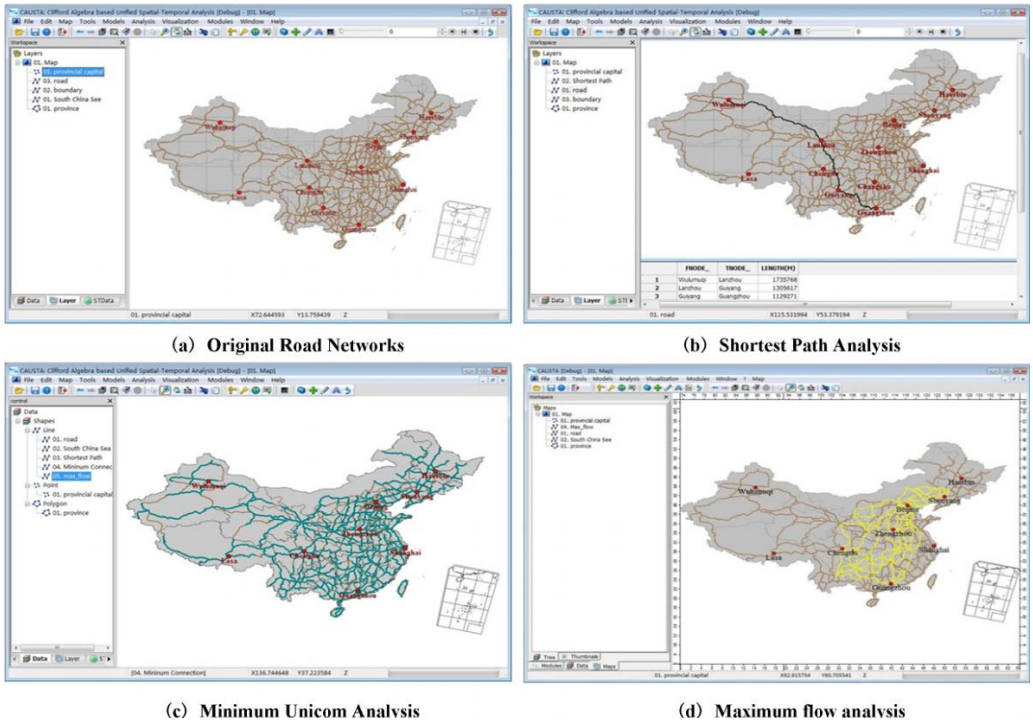



**Figure 3. Network Analysis of China Highway Road Network**

## 4. HIGH-DIM VORONOI DIAGRAM

Delaunay triangulation and Voronoi diagram can be applied in many data organization and analysis tasks (e.g. [Sam06]). Most of their implementations are typically for 2-d objects. For higher dimensional data, performance efficiency, flexibility, and extendibility of data structures are more complex. Several higher dimensional Voronoi analysis algorithms have been proposed under Clifford Algebra framework (e.g, [Zon07], [Dor07]). However, existing algorithms are not very perfectly suited for GIS data. Here, we use [Zon07]'s programming framework but define data structures of our own except using CGAL, which may provide more extensibility for GIS spatial data. The test of our algorithm can be seen in figure 4. See [Yua10] for details of the algorithms characters.

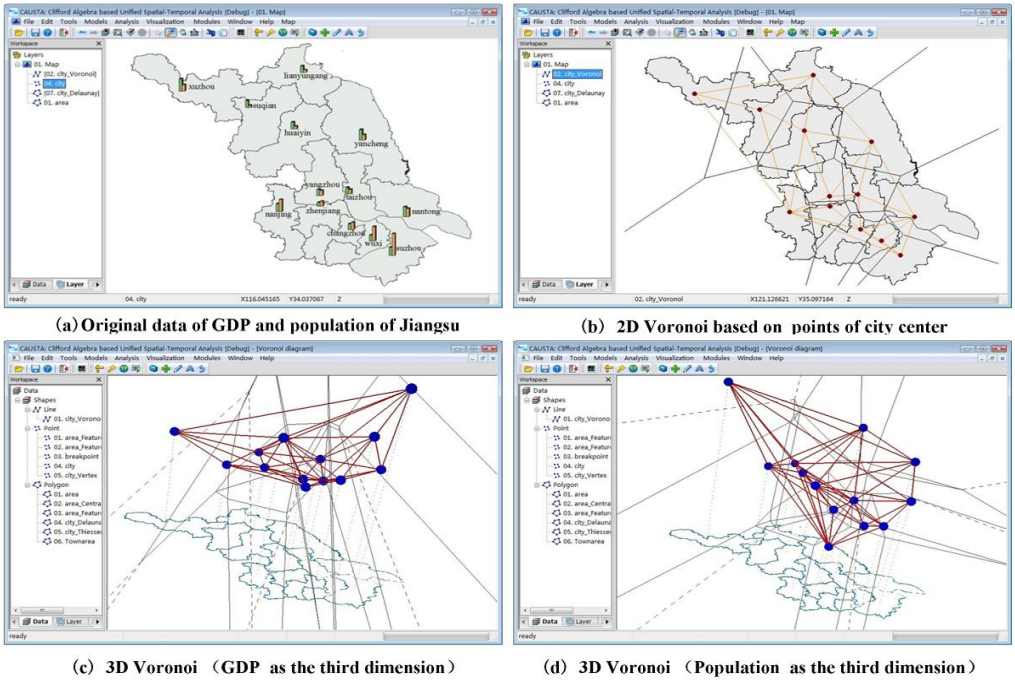



**Figure 4. Voronoi Diagram based on Clifford algebra (Adapted from [Yua10])**

## 5. CONCLUSIONS

In this article, we introduced Clifford algebra into multi-dimensional GIS spatial analysis. We proposed a unified data model that can represent multi-dimensional GIS data as standalone data objects in multivector form. Two kinds of GIS analysis algorithms: the geographical network analysis algorithm and the high dimensional Voronoi diagram were then proposed. Test on different kinds of geographical data suggest both the correctness and performance of our algorithms. All of the above suggest Clifford algebra can be seen as a new, powerful tool for multidimensional GIS analysis.

## 6. ACKKNOWLEDGMENTS

Thanks for support by the National High Technology R&D Program of China (Grant no. 2009AA12Z205) and Key Project of National Natural Science Foundation of China (Grant no. 40730527).

## 7. REFERENCES

[Lon05] Longley P A, et al. Geographic Information Systemsand Science (Second edn). John Wiley and Sons, 2005

[Yua09] Yuan M. Knowledge discovery of geographic dynamics in spatiotemporal data. In Miller H J and Han J (eds) Geographic Data Mining and Knowledge Discovery, Second edn. Taylor and Francis, pp. 347–366, 2009

[Peu95] Peuquet, D. J. and Duan, N. An Event-based Spatio-Temporal Data Model (ESTDM) for temporal analysis of geographical data. Int J Geogr Inf Sci. vol. 9, pp 7–24, 1995

[Wac94] Wachowicz, M. and Healey, R.G. Towards Temporality in GIS. Taylor and Francis, 1994

[Liu08] Liu, Y., et al. Towards a general field model and its order in GIS. Int J Geogr Inf Sci. vol. 22, pp. 623–643, 2008

[Cur07] Curtin, K.M. Network analysis in geographic information science: Review, assessment, and projections. CaGIS. vol. 34, pp. 103–111, 2007

[Sch10] Schott, R. and Staples, G.S. Reductions in computational complexity using Clifford algebras. Adv. Appl. Clifford Algebr. vol. 20, pp. 121–140, 2010

[Yua10] Yuan, L.W., et al. CAUSTA: Clifford Algebra-based Unified Spatio-Temporal Analysis. Transactions in GIS. vol.14, pp. 59–83, 2010

[Sam06] Samet, H. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, 2006

[Zon07] Zonnenberg, C. Conformal Geometric Algebra Package. Thesis of Utrecht University: http://www.cs.uu.nl/groups/MG/gallery/CGAP/index.html

[Dor07] Dorst, L., Fontijne, D., and Mann, S. Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry. Morgan Kaufmann, 2007

# Linking 2D data to a 3D Architectural model

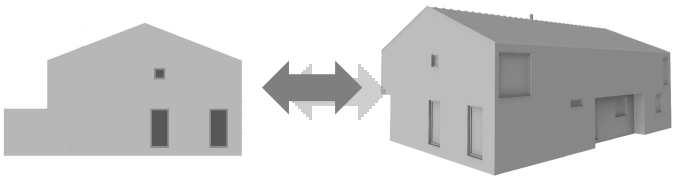
F. Gaitto Pereira
Di/ Uminho
Campus de Gualtar, 4704-553 Braga
Citar/ UCP
Rua Diogo Botelho 1327, 4169 - 005 Porto
Portugal
fgaitto@porto.ucp.pt

## ABSTRACT

In this paper we describe a method for identifying on a 3d geometry of a building, the analyses from the rendered image. The goal of this process is to link two different data types, development of a 2D analysis and link the result to the 3D model on an architectural domain.

## Keywords

Modeling, Object Retrieval, Clustering



## 1. INTRODUCTION

The common process for producing a 2d image is made from pushing 3D data to the frame buffer. By doing this, the camera defined for the render will be selective and the visible data will be processed.

If on one hand, the problem of acquiring visible data is solved, on the other hand, there is no tracking of any kind of data visible or hidden on the image. Since the goal of the render is to get the image of the model, the process works one way and forward.

Performing analyses on a 3D data of an architectural model is a task that depends on the construction of the 3D model, since there are many different methods of modeling the same object, there would be a need for different analyses approaches for each model. Nevertheless, the resulting image of the model will be the same, therefore, our process focuses on the render and links the analyses of the image to the original 3D data.



## 2. RELATED WORK IN 3D ARCHITECTURAL DOMAIN

Attribute retrieval in architectural domain is a rather unexplored field, though, depending on the goal of the work, some methods for analysis of 3D and 2D data, can be found.

Linking data from 2D or 3D is used on several papers with different goals.

A process for geometry construction is proposed by [MZW*06], that consists in rebuilding geometry according to an image analyses. The retrieval of windows of a building is performed by the detection of the frequency related to the window alignment on the image. A 3D model of the building is produced according to the retrieved 2d shapes.

One study describing shapes in the Architectural domain was developed [WBK08] using groups of horizontal slices of the model. These images are used to identify cells that represent rooms of an apartment or a house. The evaluation of cells of the image results in a connectivity graph.

In the Virtual Reality field, [DF08] developed a process to detect tracks for agents in 3D worlds. Horizontal 2D image slices are produced and floor information is fed to agents to move in the 3D model.

Linking data can be of great use and has a wide application on several fields. In our process linking data is performed to feed back information from 2D analyses into the 3d model.

3D Geometry has qualities compared to other multimedia files. In [VKF*06], is presented a synthesis of methods for analyzing geometry and the advantages of this type of data.

Detecting characteristics on a 3D architectural model like in other models, goes beyond its morphology, since they have arbitrary topologies [VKF*06]. Therefore, standard methods for shape retrieval and analyses of 3D building models aren't efficient and methods related to architecture grammar are needed. Building an architecture grammar is possible from the analyses of the render and returning the meaning to the model. Linking back to 3D data will increase the value of the 3d architectural model in any field of work.

## 3. OVER VIEW

The method starts with an OBJ file format model. Its geometry is triangulated as a standard procedure for ease of conversion from any kind of format. Triangulated OBJ models are generally used for its versatility in accessing geometry description. However, this file format has a generic data structure dedicated to all kind of 3D models, therefore, the lack of meaning or semantic orientated structure, makes it similar to a polygon soup.

From the OBJ loaded model, several polygon lists and respective renders are created.

The method is developed from an original list of polygons and by refining this list into others, based on the analyses of the new rendered image.

Each time a new list is refined, its sent to the frame buffer. This is made in several stages until one image with architectural features is obtained.

The link is possible by performing any analyses on the final image and tracking to the respective polygon through the image with color tags.

A particular pixel position in one image has different meaning on another image but relates to the same polygon.

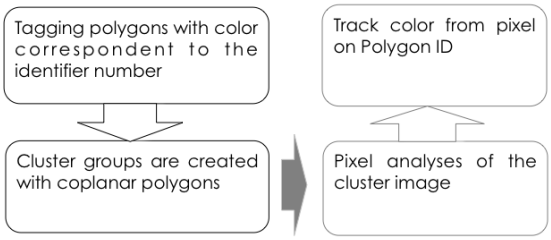



Figure 1. Setup workflow

## 4. POLYGON TAGGING

In the beginning of this stage, we need to produce an image with visible polygons from the CamList. Before the render process, each polygon is tagged with an exclusive ID color per vertex, according o each polygon identifier.

Since the tagging process uses sequential integers from the face ID to correspond the same RGB integer, color similarity currently prevented an adequate identification of the face, therefore in the tracking stage, to prevent color mismatch on sequential RGB colors, each ID color is multiplied by 10. This procedure ensures a good contrast by breaking the sequential color chain and later, the colors identified are easily reverted to the true polygon ID.

A 32-bit integer is used. All polygons will be sent to the frame buffer with their respective identifier referred previously. Each polygon is projected according to length distance, this way, the polygons close to the camera will be the last to be sent to the buffer and will not became hidden by others.

The Polygon Tagging stage, results on an image with all triangle faces colored but without any visual meaning.

A tracing algorithm identifies each colors in the image allowing to select the correspondent polygons from the CamList. A new list, TagList, is created for all visible triangles of the first image.

This image produced from the TagList is the key image of the linking process, since it will be used two ways: forward to the cluster production and backwards to track the polygon faces.

This process of color tagging, establishes the link between the two worlds, the 2D image and the 3D geometry.

The TagList will be reconstructed according to depth clusters and another image is produced. This procedure will be presented further on.

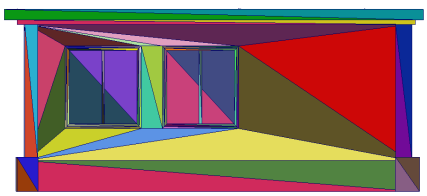


Figure 2. Buffer image of tagged triangles (for better identification purposes, colours are simulated)

## 5. CLUSTER CREATION

The Cluster creation is performed by grouping triangles of a certain type. The goal is to achieve an image with architectural meaning based on the image produced by the TagList using the only information of the vertex of the OBJ file.

A polygon face is offered on the image through the render, but the depth information has no graphical presentation on the image. Our process, is meant to include this information by the form of a gray scale color.

All triangles with the same depth information are grouped under a new list and render image. From the TagList a new list is made by placing in order of proximity all triangles.

This new list, clusterList, that produces the clusterImage, gathers groups of coplanar triangles according to their depth value and colors each set of polygon face in grey scale according to de group they belong.

Since modeling sometimes is not an accurate task, a small tolerance allows to include on the same cluster, faces that have vertex out of a particular depth.

The inquiry is made by checking the distance from the reference camera plane, up to each vertex on the TagList.

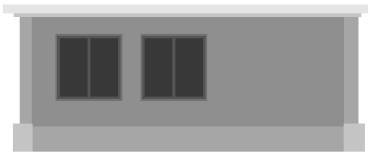


Figure 3. Image of Clustered triangles (for better identification purposes, colours are simulated)

This process revealed more efficient than others usual processes [KT03], since the groups of triangles are made from pre selected geometry and using the exclusively information from the image.

The coloring of the faces is destined not only to differentiate depth levels but to represent the distance between each plane.

Other processes of depth identification based on ZBuffer, are used in studies were the need for depth precision is not crucial.

A precise gray value correspondent to de distance, allows to perform analysis on an image with 3D information.

This paper, is meant to establish a process for linking data therefore, no analysis procedure is presented.

## 6. LINKING DATA

The clean cluster image allows performing many types of image analyses and retrieving architectural features is simplified.

Acquiring areas of interest can be tracked back to the model by matching the pixels to the same ones on the TagImage.

Starting on a pixel or group of pixels on the cluster image (pixels selected or resultant from any type of analysis), the position is checked on the previous image, the TagImage, in the same pixel position. This inquiry informs the RGB color of the pixel and after cutting back the increment used on the TagImage building, it lets us know the true polygon ID.

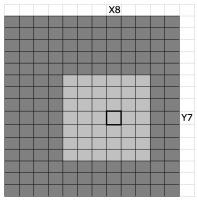



Figure 4. Pixel on Clustered Image

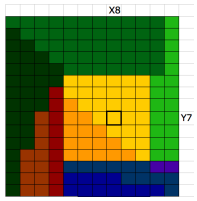



Figure 5. Pixel on TagImage

## 7. CONCLUSION AND FUTURE WORK

This paper presents a simple and practical method for linking 2D to 3D data.

The colors from the referenced pixel or the area of interest, point directly to the polygons of the model. Clustering polygons on the model, allows restructuring the 3D architectural model.

From the experimental results we can draw the conclusion that this method is reliable and useful for a wide range of analyses on 3D architectural models.

The current application developed on C#, besides using OBJ files, isn't meant for performance.

However heavy loaded models are simplified at the start due to visibility selection.

Models of buildings usually are modeled from the outside for exterior render, however, they can also contain interior geometry either to represent inside divisions or simply to represent the thickness of the exterior wall.

This process allows to have a selective approach by filtering the exterior geometry independent from the modeling of the building.

Therefore, complete models with inside and outside geometry and about 250000 polygons where easily loaded and rendered.

Nevertheless, smaller polygon models can have the same performance as bigger ones, since only the visible geometry is processed.

In future work this method is to be extended to retrieval of architectonic attributes and restructuring of the 3D model, such as geometry coloring of elements

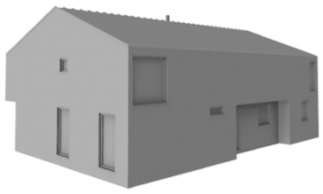


Figure 6. architectural model including interior geometry

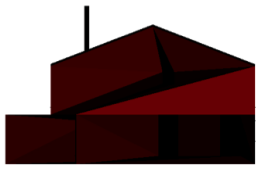


Figure 7. TagImage of East elevation

Figure 8. ClusterImage of East elevation

## 8. ACKNOWLEDGMENTS

Our thanks António Ramires Fernandes, UCP CITAR, António oliveira, Diogo Ferreira, for their guidance, advice and support.

## 9. REFERENCES

[MZW*06] Pascal Mueller, Gang Zeng, Peter Wonka and Luc Van Gool ”*Image-based Procedural Modeling of Facades*” Proceedings of ACM SIGGRAPH 2007 / ACM Transactions on Graphics

[WBK08] R. Wessel, I. Blümel, R. Klein, The Room Connectivity Graph: Shape Retrieval in the Architectural Domain. In WSCG 2008

[DF08] L. Deusdado, O. Belo, A. Fernandes. Pedestrian behavioral simulation in real 3D environments, The 11th International Conference on Computer Graphics and Artificial Intelligence, Athens (GREECE), May 30-31, 2008.

[VKF*06] Golovinskiy, Vladimir G. Kim, Thomas Funkhouser, Shape-based Recognition of 3D Point Clouds in Urban Environments, International Conference on Computer Vision(ICCV)September 2009

[KT03] S. Katz, A. Tal, Hierarchical Mesh Decomposition Using Fuzzy Cluster and Cuts, ACM 2003

# A Note on Geometric Algebra and Neural Networks

*Kanta TACHIBANA
kanta@cc.kogakuin.ac.jp

MinhTuan PHAM
minhtuan@cmplx.cse.nagoya-u.ac.jp

Tomohiro YOSHIKAWA
{yoshikawa

Takeshi FURUHASHI
furuhashi}@cse.nagoya-u.ac.jp

**Abstract:** This note first explains Clifford's geometric algebra (GA) as a generalization of complex and quaternion algebras. Second, this note describes GA neurons as a natural extension of complex neurons. In any dimension the GA neuron takes a vector input and returns another vector output. The GA neurons are applicable to optimization of Space Folding Model for effective pattern recognition. Next, points with precision are considered using conformal geometric algebra and it is shown that addition of conformal vectors works well for precision update. The GA neuron and its use of vectors with precision (or belief) could be useful for datasets with different levels of precision/detail/belief of any dimension. The conformal vector could be also useful to set a prior distribution of geometric versors.

## 1 Introduction

Let $T(L)$ be the tensor space of the linear vector space $L$. Grassmann's exterior algebra $E(L)$ regards all elements of $T(L)$ which contain tensor products of any $x \in L$ with $x$ itself as the zero element of $T(L)$. The exterior product or wedge product maps an ordered pair $E(L) \times E(L)$ to $E(L)$. The exterior product is bilinear and $x \wedge x$ becomes 0 for any $x$ in $L$. Because $0 = (x + y) \wedge (x + y) = x \wedge x + x \wedge y + y \wedge x + y \wedge y$ where $x$ and $y$ are both elements of $L$, the exterior product is anti-commutative $x \wedge y = -y \wedge x$.

Clifford's geometric algebra $G(L)$ is the exterior algebra of a linear space $L$ equipped with a measure $x \cdot x = |x|^2$. $G(L)$ has a bilinear and associative product, that maps an ordered pair $G(L) \times G(L)$ to $G(L)$, which is called geometric product or Clifford product. For $x, y \in L$, the geometric product (simply written by juxtaposition of the elements) is defined as $xy = x \cdot y + x \wedge y$. Example: Let us think about the two-dimensional Euclidean space $\mathbb{R}^2$. The geometric products of its orthonormal basis vectors $\{e_1, e_2\}$ are $e_1e_1 = e_1 \cdot e_1 = e_2e_2 = e_2 \cdot e_2 = 1$ and $e_1e_2 = e_1 \wedge e_2 = -e_2 \wedge e_1 = -e_2e_1$. Because of the associativity we have $(e_1e_2)^2 = -e_2e_1e_1e_2 = -e_2e_2 = -1$. We therefore denote the unit bivector as $i = e_1e_2$, then $i^2 = -1$. The set of real linear combinations of $\{1, i\}$ is the even grade subalgebra of $G(\mathbb{R}^2)$, which is isomorphic to the set of complex numbers $\mathbb{C}$.

The authors have naturally extended the complex-valued neuron, all of whose input, weight, bias and output are in $\mathbb{C}$ [1, 2]. The proposed neuron uses the so-called Clifford group $= \{s \in G(L) \mid \varphi(s, x) \in L \ \forall x \in L\}$, where $\varphi$ is a function constructed with geometric products, with weight in $G(L)$, and with input, output and bias in $L$ [3]. This note discusses the relationship of the GA neuron with complex and quaternion neurons [4]. This note also considers points with precision using conformal geometric algebra.

## 2 Complex Neuron

The complex neuron in general sums input stimuli weighted by weights plus bias all of which are complex numbers. For simplicity, assume the neuron has an input:

$$u_{\mathbb{C}} = w_{\mathbb{C}} x_{\mathbb{C}} + b_{\mathbb{C}},$$

where $u_{\mathbb{C}}, w_{\mathbb{C}}, x_{\mathbb{C}}, b_{\mathbb{C}} \in \mathbb{C}$. A two-dimensional vector $(x_1, x_2)$ can be represented as a complex number. Its first and second components are the real and the imaginary coefficients respectively: $x_{\mathbb{C}} = x_1 + x_2 i \in \mathbb{C}$. On the other hand, a natural representation of a vector is $x = x_1e_1 + x_2e_2$. Using geometric algebra $G(\mathbb{R}^2)$, we can link it to the complex number representation as $x_{\mathbb{C}} = x_1(e_1e_1) + x_2(e_1e_2) = e_1(x_1e_1 + x_2e_2) = e_1x$. The square root of $w_{\mathbb{C}} = \rho(\cos\theta + i\sin\theta),\ \rho \in [0, \infty),\ \theta \in [0, 2\pi)$ is also a complex number $w'_{\mathbb{C}} = \sqrt{w_{\mathbb{C}}} = \sqrt{\rho}(\cos\frac{\theta}{2} + i\sin\frac{\theta}{2})$. And complex numbers are commutative, i.e. $w'_{\mathbb{C}}(e_1x) = (e_1x)w'_{\mathbb{C}}$. Then, the complex neuron becomes:

$$\begin{aligned}(e_1u) &= w'_{\mathbb{C}}w'_{\mathbb{C}}(e_1x) + (e_1b)\\ &= w'_{\mathbb{C}}(e_1x)w'_{\mathbb{C}} + (e_1b).\end{aligned}$$

Multiply $e_1$ from the left:

$$e_1e_1u = e_1\underline{w'_{\mathbb{C}}e_1}xw'_{\mathbb{C}} + e_1e_1b.$$

Looking at the underlined part, and let $w'_{\mathbb{C}} = \alpha + \beta i$,

$$\begin{aligned}w'_{\mathbb{C}}e_1 &= (\alpha + \beta i)e_1 = \alpha e_1 + \beta e_1e_2e_1\\ &= e_1(\alpha - \beta i) = e_1\overline{w'_{\mathbb{C}}},\end{aligned}$$

where $\overline{w'_{\mathbb{C}}}$ is the complex conjugation. Then, the complex neuron is represented as:

$$\begin{aligned}e_1e_1u &= e_1e_1\overline{w'_{\mathbb{C}}}xw'_{\mathbb{C}} + e_1e_1b\\ u &= \overline{w'_{\mathbb{C}}}xw'_{\mathbb{C}} + b. \qquad (1)\end{aligned}$$

This $u$ is the result of rotating $x$ by the angle $\theta$, scaling by factor $\rho$, and translation by vector $b$.

## 3 Geometric Algebra Neuron

The geometric algebra neuron is a natural extension of the complex neuron. Let the Clifford group $\Sigma =$

$\{s \in G(L) \mid \varphi(s,x) \in L \ \forall x \in L\}$, whose element transforms a vector to another vector. For $k = 0, 1, \ldots, n$, the set of versors, i.e. multiplications of $k$ linearly independent vectors $\{M_k = v_1 v_2 \ldots v_k \in G(L) \mid v_1, \ldots, v_k \in L\}$ is a subset of $\Sigma$ with $\varphi(M_k, x) = \overline{M_k} x M_k$. The authors have proposed a geometric algebra neuron:

$$u = \sum_{k=0}^{n} \varphi(M_k, x) + b$$

and found optimal learning rates based on the Hessian matrix. Note that the complex neuron of eq. (1) only represents $\varphi(M_2, x)$ part. In the case of $n = 2$, $\varphi(M_0, x)$ is scalar multiplication, $\varphi(M_1, x)$ is a reflection, and $\varphi(M_2, x)$ is a rotation. These three transformations are mixed. The mixing weights are adjusted with the norm $|M_k|^2$. In the case of $n = 3$, $\varphi(M_2, x)$ is isomorphic to the quaternion product which gives rotation and scaling in three-dimensional space.

As the GA neuron learns reflection and rotation of vectors and multivectors in any dimension, a network constructed with GA neurons can be applicable to optimization of Space Folding Model (SFM) [5]. In the network for SFM, a GA neuron is assigned for each Space Folding Vector (SFV) and the feature space is folded to minimize the error function by training $\{M_k\}s$.

## 4 Conformal GA and Update of Precision

Introducing new two basis vectors $\vec{e}_+$ and $\vec{e}_-$ to $G(\mathbb{R}^n) = G(n)$, we have conformal geometric algebra $G(n+1, 1)$. The new vectors have positive and negative signature, i.e. $\vec{e}_+^{\,2} = -\vec{e}_-^{\,2} = 1$. And further new basis vectors are constructed in the $\vec{e}_+ \wedge \vec{e}_-$ plane:

$$\begin{cases} \vec{n}_\infty = \vec{e}_+ + \vec{e}_- \\ \vec{n}_o = \frac{1}{2}\left(\vec{e}_- - \vec{e}_+\right). \end{cases}$$

Because $\vec{n}_\infty^2 = \vec{n}_o^2 = 0$, they are also called null basis vectors. Using these null basis vectors, $n$-dimensional hypersphere with center at $\vec{x} \in \mathbb{R}^n$ and radius $\rho \in \mathbb{R}$ is represented as

$$X = \mu\vec{x} + \frac{\mu}{2}\left(\vec{x}^2 - \rho^2\right)\vec{n}_\infty + \mu\vec{n}_o,$$

where $\mu$ is any nonzero real number.

We regard $\rho^2$ of the conformal vector as precision, say $\rho^2 = \beta = \sigma^{-2}$, where $\sigma$ represents a standard deviation. In this interpretation, addition of two conformal vectors means:

$$\begin{aligned} X + Y &= \vec{x} + \vec{y} + \frac{1}{2}\left(\vec{x}^2 - \beta_x + \vec{y}^2 - \beta_y\right)\vec{n}_\infty + 2\vec{n}_o \\ &\propto \frac{\vec{x}+\vec{y}}{2} + \frac{1}{2}\left(\frac{\vec{x}^2+\vec{y}^2}{2} - \frac{\beta_x+\beta_y}{2}\right)\vec{n}_\infty + \vec{n}_o \\ &= \vec{m} + \frac{1}{2}\left(\vec{m}^2 - \beta\right)\vec{n}_\infty + \vec{n}_o, \end{aligned}$$

where $\vec{m} = (\vec{x}+\vec{y})/2$, i.e. the midpoint, and $\beta = (\beta_x+\beta_y)/2 - \left\{(\vec{x}-\vec{m})^2 + (\vec{y}-\vec{m})^2\right\}/2$. The new precision $\beta$ is interpreted as average precision minus variance.

This can be generalized to weighted points. Let $\mu_x \in \mathbb{R}$ be the weight.

$$X = \mu_x\vec{x} + \frac{\mu_x}{2}\left(\vec{x}^2 - \beta_x\right)\vec{n}_\infty + \mu_x\vec{n}_o.$$

The addition of weighted points is:

$$\begin{aligned} X + Y &= \mu_x\vec{x} + \mu_y\vec{y} + \frac{\mu_x(\vec{x}^2 - \beta_x) + \mu_y(\vec{y}^2 - \beta_y)}{2}\vec{n}_\infty \\ &+ (\mu_x + \mu_y)\vec{n}_o \\ &= \mu\vec{m} + \frac{\mu}{2}\left(\vec{m}^2 - \beta\right)\vec{n}_\infty + \mu\vec{n}_o, \end{aligned}$$

where $\mu = \mu_x + \mu_y$ is the new weight and

$$\begin{aligned} \vec{m} &= \frac{\mu_x\vec{x} + \mu_y\vec{y}}{\mu} \\ \beta &= \frac{\mu_x\beta_x + \mu_y\beta_y}{\mu} - \frac{\mu_x(\vec{x}-\vec{m})^2 + \mu_y(\vec{y}-\vec{m})^2}{\mu} \end{aligned}$$

$\vec{m}$ is the internally dividing point (center of mass) and the new precision $\beta$ is interpreted as weighted average precision minus weighted variance.

This fact of good update of precision can be useful in the following cases.

1. Each training/test sample has a different level of precision $\beta$ (or belief).
2. Massive samples must be learnt and coarse graining is effective. Here, a lot of samples are learnt at once as a hypersphere sample.
3. Precision (or belief) characterises a prior distribution of a dataset and neuron parameters.

## 5 Conclusion

This note described the GA neuron as a natural extension of the complex neuron. In any dimension the GA neuron inputs a vector and outputs another vector. Next, points with precision were considered using conformal geometric algebra. And we showed that the addition of conformal vectors works well to update precision. We want to note the possibility of combining GA neuron with conformal representation of precision in the analysis/learning of datasets with various levels of details and in the coarse graining of huge datasets. Bayesian updates of weights could also be represented by conformal vectors.

## acknowledgment

The authors thank Dr. Eckhard Hitzer for his suggestions.

## References

[1] S. Buchholz, K. Tachibana, E. Hitzer, Optimal Learning Rates for Clifford Neurons, in J. Marques de Sa et al. (Eds.): Proceedings of ICANN 2007, Part I, LNCS 4668, Springer-Verlag Berlin 2007, pp. 864–873.

[2] S. Buchholz, E. Hitzer, K. Tachibana, Coordinate independent update formulas for versor Clifford neurons, SCIS and ISIS 2008, Nagoya, September 2008.

[3] P. Lounesto, Clifford Algebras and Spinors, Cambridge University Press, 2001

[4] T. Nitta (ed.), Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters, Information Science Publishing, 2009.

[5] M. T. Pham, T. Yoshikawa, T. Furuhashi, Pattern Recognition Based on Space Folding Model, Proc. Applied Geometric Algebras in Computer Science and Engineering (AGACSE2010), June 2010.