International Workshop on

Computer Graphics, Computer Vision and Mathematics

in co-operation with

EUROGRAPHICS

GraVisMa 2009

Workshop Proceedings

Edited by

Vaclav Skala, University of West Bohemia, Czech Republic
Dietmar Hildenbrand, Technical University Darmstadt, Germany

Vaclav Skala – Union Agency

GraVisMa 2009 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 978-80-86943-90-9*

Foreword

The 1st international workshop on ***Computer Graphics, Computer Vision and Mathematics*** was held at the University of West Bohemia in Plzen (Pilsen), Czech Republic on September 2-4, 2009. Informal atmosphere of the GraVisMa 2009 workshop stimulated scientific discussions between researchers and practitioners that will hopefully lead to further research international collaborations and common project proposals as well.

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

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

GraVisMa workshops brings new impulses to related fields of computer science, especially in development of new approaches and algorithms and will stimulate research activities between mathematicians and computer science experts.

GraVisMa workshop proceedings contain full papers and communication papers presented at the workshop that were commented, reviewed by participants and external reviewers. The papers accepted for publication were significantly improved after the workshop due to those discussions and comments.

GraVisMa 2009 attendees

There were **64** papers submitted, for publication in the proceedings **15** FULL (23%) and **9** Communication (14%) papers were accepted. We had also keynotes:

- Eckhard M. S. Hitzer: Fourier and Wavelet Transformations in Geometric Algebra, Fukui University, Japan
- Werner Benger **et al:** Using Geometric Algebra for Navigation in Riemannian and Hard Disc Space, Louisiana State University, USA
- Anthony Lasenby **et al:** Rigid Body Motion and Conformal Geometric Algebra, Cambridge University, U.K.
- Leo Dorst: Conformal Geometric Algebra by Extended Vahlen Matrices, University of Amsterdam, The Netherlands

The Co-Chairs would like to thanks to all who:

- contributed to this workshop, especially to all commented and / or reviewed papers,
- helped this workshop to be held and to people that helped during the workshop itself.

It is on a reader to evaluate how the GraVisMa goals and intentions have been fulfilled by the GraVisMa 2009 workshop.

Co-Chairs

Vaclav Skala
University of West Bohemia
Czech Republic

Dietmar Hildenbrand
Technical University Darmstadt
Germany

List of people who commented / reviewed presentations and papers

Contents

Geometric Algebra Computers

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

ABSTRACT

Geometric algebra covers a lot of other mathematical systems like vector algebra, complex numbers, Plücker coordinates, quaternions etc. and it is geometrically intuitive to work with. Furthermore there is a lot of potential for optimization and parallelization.
In this paper, we investigate computers suitable for geometric algebra algorithms. While these *geometric algebra computers* are working in parallel, the algorithms can be described on a high level without thinking about how to parallelize them. In this context two recent developments are important. On one hand, there is a recent development of geometric algebra to an easy handling of engineering applications, especially in computer graphics, computer vision and robotics. On the other hand, there is a recent development of computer platforms from single processors to parallel computing platforms which are able to handle the high dimensional multivectors of geometric algebra in a better way.
We present our geometric algebra compilation approach for current and future hardware platforms like reconfigurable hardware, multi-core architectures as well as modern GPGPUs.

Keywords: Geometric algebra, GPGPU, multi-core-architecture, Verilog, OpenCL, CUDA, OpenMP, Ct, Larrabee.

1 INTRODUCTION

The foundation of geometric algebra was laid in 1844 and 1862 by Hermann Grassmann [8] whose 200th birthday we were celebrating in 2009 [25]. His work was continued by the English mathematician W. K. Clifford in 1878 [2]. Due to the early death of Clifford, the vector analysis of Gibbs and Heaviside dominated most of the 20th century, and not the geometric algebra. Geometric algebra has found its way into many areas of science, since David Hestenes treated the subject in the '60s [9]. In particular, his aim was to find a unified language for mathematics, and he went about to show the advantages that could be gained by using geometric algebra in many areas of physics and geometry [12], [10], [13] culminating in the development of the conformal geometric algebra [11]. Many other researchers followed and showed that applying geometric algebra in their field of research can be advantageous, e.g. in engineering areas like computer graphics, computer vision and robotics. Please find a survey on geometric algebra algorithms in [26].

During the past decades, especially from 1986 until 2002, processor performance doubled every 18 months. Currently, this improvement law is no longer valid because of technical limitations. Now, we can recognize a shift to parallel systems and most likely these systems will dominate the future. Thanks to multi-core architectures or powerful graphics boards for instance based on the CUDA technology from NVIDIA or on the future Larrabee technology of INTEL, one can expect impressive results using the powerful language of geometric algebra.

There is already a very advanced pure software solution called Gaigen (see [4] and [5]) as well as some pure hardware solutions geometric algebra algorithms (see for instance [24], [21] and [7] and a survey in [17]).

We propose to combine the advantages of both software and hardware solutions. We use a two-stage compilation approach for geometric algebra algorithms. In a first step we optimize geometric algebra algorithms with the help of symbolic computing. This kind of optimization results in very basic algorithms leading to highly efficient software implementations. These algorithms, foster a high degree of parallelization which are then used for hardware optimizations in a second step. As examples for geometric algebra computing we present

- a FPGA(field programmable gate array) implementation of an inverse kinematics algorithm.
- examples on how to implement geometric algebra algorithms on multi-core architectures. Since all of the coefficients of high dimensional multivectors can be computed in parallel, geometric algebra computing benefits a lot from highly parallel structures.
- a OpenCL/CUDA implementation for arbitrary geometric products using 2^n-dimensional multivectors of n-dimensional geometric algebras.

2 GEOMETRIC ALGEBRA COMPUTING APPROACH

Geometric algebra offers some very interesting properties like

- it is geometrically intuitive to work with
- it is easy to handle geometric objects like spheres, circles, planes etc. as well as geometric operations like rotations, reflections etc.

- geometric algebra algorithms are very compact
- it covers a lot of other mathematical systems like vector algebra, complex numbers, Plücker coordinates, quaternions etc.

How can we combine these properties with highly performant implementations? Multivectors of a n-dimensional geometric algebra are 2^n-dimensional. At first glance, this seems to be computationally very expensive. But, there is a lot of potential for optimization and parallelization of multivectors:

- the possibility of precomputing geometric algebra expressions
  - determine which of the coefficients are needed for the resulting multivector
  - symbolic simplification of the remaining coefficient computations
- Since all of the remaining coefficients can be computed in parallel, geometric algebra computations benefit a lot from parallel structures.

This is why we propose to separate geometric algebra computing in two layers

- geometric algebra (GA) compilation layer
- platform layer

At the GA compilation layer geometric algebra operations like geometric product, outer product, inner product, dual and reverse on multivectors are handled. This is compiled in a second step to the platform layer. On this layer only basic arithmetic operations on multivectors with a high potential for efficient computations on parallel platforms are available.

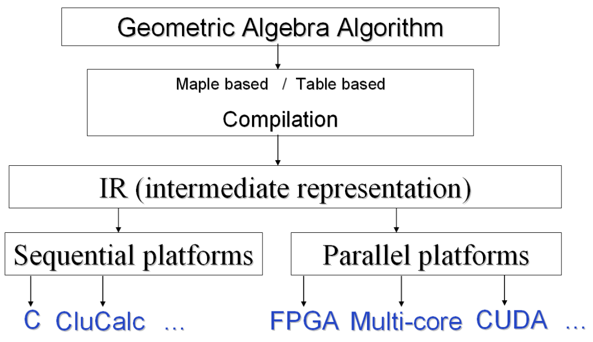

Geometric Algebra Algorithm
Maple based / Table based
Compilation
IR (intermediate representation)
Sequential platforms
Parallel platforms
C
CluCalc
...
FPGA
Multi-core
CUDA
...

Figure 1: Geometric algebra computing architecture. Algorithms are compiled to an intermediate representation for the compilation to different computing platforms.

Our geometric algebra computing architecture is presented in Figure 1. Algorithms (described by the geometric algebra programming language CLUCalc [23]) are compiled to an intermediate representation using a Maple based or a table based approach (see sections 3.1 and 3.2). Based on this representation implementations for different sequential and parallel platforms can be derived. See some examples for geometric algebra computer platforms based on FPGA- Multicore- and OpenCL/CUDA-architecture in sections 4.1 to 4.5.

3 GEOMETRIC ALGEBRA COMPILATION

In order to achieve highly efficient implementations, geometric algebra algorithms have to be optimized first. We use two different compilation approaches. The Maple based compilation needs the commercial Maple package and is restricted to geometric algebras with dimension $<= 9$. The table based compilation is able to handle higher dimensional algebras but it is currently not as powerful as the Maple based compilation.

3.1 Maple Based Compilation

The Maple based compilation uses the powerful symbolic computation feature of Maple [20]. Since all of the results of geometric algebra operations on multivectors are again multivectors we symbolically compute and simplify the resulting multivectors in order to determine which of the coefficients are actually needed and what is the most simple expression for each coefficient (in the Maple sense).

There is already a first implementation of a compiler for geometric algebra algorithms called Gaalop (Geometric algebra algorithms optimizer) working with this approach. Please find some information in [17]. You are able to download Gaalop from [16].

3.2 Table Based Compilation

The table based compilation approach uses precomputed multiplication tables inspired by the code generator Gaigen [6] from the university of Amsterdam. While Gaigen needs explicit specialization of multivectors this is done automatically in our approach (see the example below).

Multiplication tables In order to compute geometric algebra algorithms, the rules for the computation of the products of multivectors have to be known. These products of specific geometric algebras can be summarized (and precomputed) in multiplication tables describing the product of different blades of the algebra. You can find some examples of multiplication tables in the appendix. Table 1, for instance, describes the geometric product of the $8 = 2^3$ blades of the 3D Euclidean geometric algebra. Based on this information the geometric product of two multivectors, each defined as a linear combination of all the blades $mv = \sum mv_i E_i$ can be easily derived as described in the caption of Table 1.

The same procedure can be used for other products. Table 2, for instance, describes the outer product of the

mv_1	mv_2	mv_3	mv_4	mv_5	mv_6	mv_7	mv_8
E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}

3D Euclidean geometric algebra. Note that a lot of entries are zero corresponding to the outer product of two identical blades.

Example Let us compile the following CLUCalc script step by step:

```
a=a1*e1+a2*e2+a3*e3;
b=b1*e1+b2*e2+b3*e3;
?c=a*b;
d=a+c;
?f=a^d;
```

It computes the geometric product of two 3D vectors, adds two multivectors and computes the outer product of two multivectors.

The first two lines are used for the definition as well as for an automatic specialization of the two multivectors a

1	2	3	4	5	6	7	8
	a_1	a_2	a_3				

and b

1	2	3	4	5	6	7	8
	b_1	b_2	b_3				

For both, only the entries 2, 3 and 4 are needed since they correspond to the three basis vectors e_1, e_2, e_3 (see Table 1).

The question mark in the third line of the CLUCalc script indicates an explicit evaluation of this line, the geometric product of the two multivectors a and b. Table 3 shows the corresponding multiplication table for this product. It is derived from the Table 1 with empty rows and columns for multivector entries not needed for a and b. The resulting multivector c needs only the coefficients for the blades E_1, E_5, E_6, E_7 (see Table 3).

```
c[1]=a1*b1+a2*b2+a3*b3;
c[5]=a1*b2-a2*b1;
c[6]=a2*b3-a3*b2;
c[7]=a1*b3-a3*b1;
```

Each coefficient $c[k]$ can be computed by summing up the products $\pm a_i * b_j$ based on the E_k table entries, for instance $c_1 = a_1 * b_1 + a_2 * b_2 + a_3 * b_3$.

1	2	3	4	5	6	7	8
$c[1]$	a_1	a_2	a_3	$c[5]$	$c[6]$	$c[7]$	

In the fourth line of the CLUCalc script, two multivectors are added resulting in the following multivector d:

The evaluation of the outer product of a with this just computed multivector d leads to

```
f[2]=a1*c[1];
f[3]=a2*c[1];
f[4]=a3*c[1];
f[5]=a1*a2-a2*a1;
f[6]=a2*a3-a3*a2;
f[7]=a1*a3-a3*a1;
f[8]=-a2*c[7]+a3*c[5]+a1*c[6];
```

For this computation you can use the multiplication table 2. Associating the rows with the multivector a and the columns with d we are able to set the rows 1, 5, 6, 7, 8 as well as the column 8 to zero. We recognize that the remaining entries are for the coefficients 2, 3, 4, 5, 6, 7 and 8, E_2 for instance in the second row and the first column associated with the product $a_1 * c[1]$.

Note that the multivector entries 5, 6 and 7 lead to zero entries. This can be either recognized at compile time or at runtime. In both cases the resulting multivector f has the following form:

1	2	3	4	5	6	7	8
	$f[2]$	$f[3]$	$f[4]$				$f[8]$

4 GEOMETRIC ALGEBRA COMPUTERS

Here, computers suitable for geometric algebra algorithms, are called *geometric algebra computers* (GA computers).

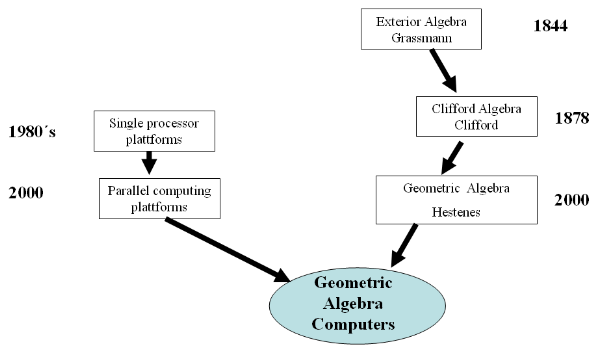

Exterior Algebra Grassmann
1844
Clifford Algebra Clifford
1878
1980´s
Single processor plattforms
2000
Parallel computing plattforms
Geometric Algebra Hestenes
2000
Geometric Algebra Computers

Figure 2: The mathematical development to geometric algebra and the computer development to parallel computing platforms leading to GA computers

There are mainly two recent developments leading to GA computers (see Figure 2):

- the development of mathematics from Grassmann´s exterior algebra to Clifford´s algebra to the geometric algebra of David Hestenes and especially the 5D conformal model leading to a lot of applications for instance in computer graphics, computer vision and robotics.
- the recent development of computer platforms for the mass market from single processors to parallel computing platforms which are able to handle the high dimensional multivectors of geometric algebra in a better way.

Figure 3 shows one example of an architecure able to compute the coefficients of a multivector in parallel.

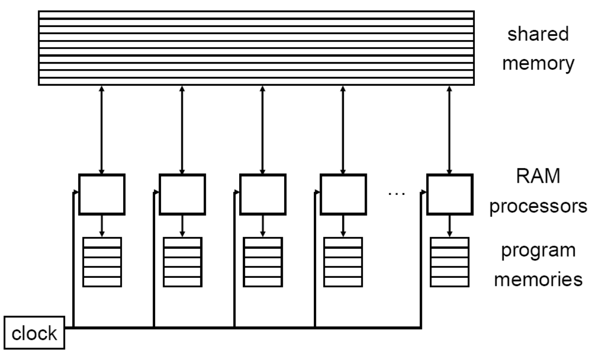


Figure 3: Computing architecture with a number of parallel processors, each consisting of local program memory. All the processors are able to communicate via global shared memory.

With the compilation approaches described in sections 3.1 and 3.2, geometric algebra algorithms are compiled into a description suitable for parallel computer platforms. In a next compilation step, the different platforms require different descriptions for their specific architecture. As follows, we describe solutions for a reconfigurable hardware implementation using Verilog, multi-core architectures using OpenMP and Ct as well as a GPGPU implementation using OpenCL/CUDA.

4.1 FPGA/Verilog implementation of a geometric algebra algorithm

There are general FPGA (field programmable gate arrays) implementations for geometric products ([24] and [7]). Our approach differs from these general solutions as we compile geometric algebra algorithms first into simplified algorithms that can be handled easily by FPGA´s. This is why we are not so much restricted in the length of the expressions to compute as well as in the dimension of the algebra.

Our FPGA implementations are always application specific. As one proof-of-concept for our approach we implemented an inverse kinematics algorithm. First, we used our Maple based compilation approach (see section 3.1) and the software implementation of the optimized algorithm became three times faster than the conventional solution [14]. The FPGA implementation of the optimized algorithm used the Verilog programming language. See Figure 4 for the data flow and the pipeline schedule of the computation of the following part of the algorithm (one coefficient of one multivector)

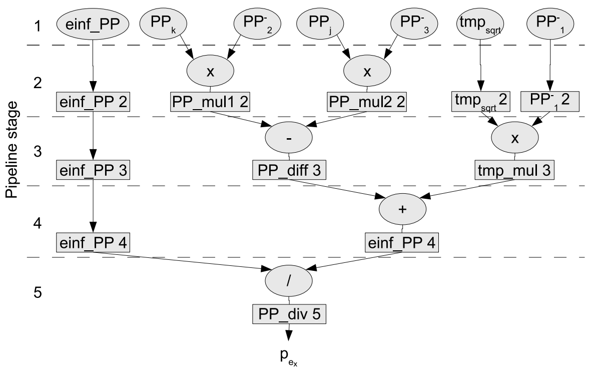


Figure 4: Pipeline schedule for the coefficient p_{ex} of a multivector. All the computations according to equation (1) of all the pipeline stages can be done in parallel.

$$p_{ex} = (PP_j(PP_{34} - PP_{35}) + PP_k(PP_{25} - PP_{24} \qquad (1)$$
$$+ tmp_{sqrt}(PP_{15} - PP_{14}))/einf\_PP.$$

This implementation became about 300 times faster [15] (3 times by software optimization and 100 times by additional hardware optimization). The main advantage of this kind of implementation on reconfigurable hardware is that we are able to realize parallelism in two dimensions

- compute all the coefficients of one (or more) multivectors in parallel
- use the pipeline structure (computations in all pipeline stages at the same time).

4.2 OpenMP

OpenMP can be used in order to parallelize GA algorithms. The programming language C can be extended with OpenMP directives for an incremental approach to parallelizing code. For details on OpenMP, please refer to [1].

OpenMP supports task parallel computations. The data of all the different threads is shared by default. This is why the coefficients of multivectors can be computed in parallel (as well as independent multivectors). Using OpenMP for C, our above mentioned example looks as follows

```
#pragma omp parallel {
    #pragma omp sections {
      #pragma omp section
      c[1]=a1*b1+a2*b2+a3*b3;
      #pragma omp section
      c[5]=a1*b2-a2*b1;
      #pragma omp section
      c[6]=a2*b3-a3*b2;
      #pragma omp section
      c[7]=a1*b3-a3*b1;
    }/*End of sections block */

    #pragma omp sections
    {
      #pragma omp section
      f[2]=a1*c1;
      #pragma omp section
      f[3]=a2*c[1];
      #pragma omp section
      f[4]=a3*c[1];
      #pragma omp section
      f[5]=a1*a2-a2*a1;
      #pragma omp section
      f[6]=a2*a3-a3*a2;
      #pragma omp section
      f[7]=a1*a3-a3*a1;
      #pragma omp section
      f[8]=-a2*c[7]+a3*c[5]+a1*c[6];

    }/*End of sections block */

  } /*End of parallel region */
```

Each of the two multivectors *c* and *f* have to be computed sequentially because *f* needs the result of *c* for its computation (while all of their coefficients can be computed in parallel). In case of no dependance of the computations, multivectors can also be computed in parallel.

4.3 Ct

Intel researchers are developing Ct, or C/C++ for Throughput Computing [18] in order to support their new multi-core platform (code name Larrabee).

Ct offers parallelism on so-called *indexed vectors* suitable for sparse multivectors. The fist step of our example of section 3 generates a multivector which can be described as the following indexed vector

```
c= [(1 -> a1*b1+a2*b2+a3*b3),
     (5 -> a1*b2-a2*b1),
     (6 -> a2*b3-a3*b2),
     (7 -> a1*b3-a3*b1),
     (_ -> 0)
   ]
```

Note that the underscore denotes a default value for empty coefficients.

All operators on indexed vectors are implicitly parallel. This is why the addition of multivectors of our example

```
d=a+c;
```

can be done very easily in Ct.

4.4 ATI stream

The ATI stream technology combines multiple thread computing with parallel computing within the threads. The following sample code computes the geometric product of the above example with the help of float4 vectors. The 4 computations for the coefficients x,y,z,w are computed in parallel.

```
kernel void MV (float4 a<>,
                float4 b<>,
                out float4 c<>){
  float4 result;
  result.x=a.x*b.x+a.y*b.y+a.z*b.z;
  result.y=a.x*b2-a2*b.x;
  result.z=a.y*b.z-a.z*b.y;
  result.w=a.x*b.z-a.z*b.x;
  c=result;
}
```

Please find an investigation about a ray tracing application using this technology in [3].

4.5 OpenCL/CUDA implementation of arbitrary geometric products

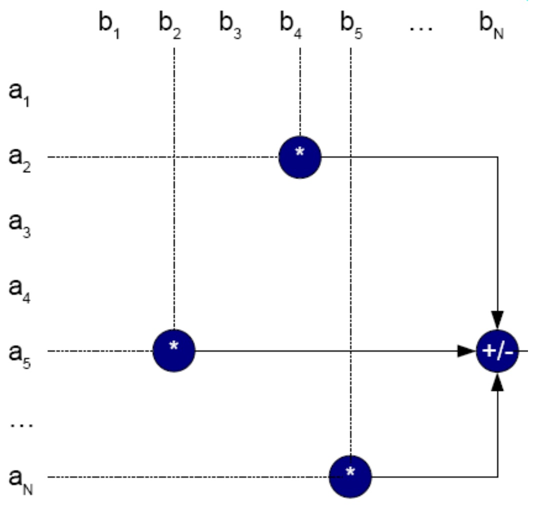


Figure 5: The result of a product of two multivectors a, b is again a multivector. Each of its coefficients is a sum of (signed or unsigned) products of coefficients of a and b.

OpenCL [19] is an open standard for parallel programming of heterogeneous systems. It is inspired by Nvidia´s CUDA technology [22]. Both, are supporting multiple threads which are able to run the same code with different data on many parallel processors.

The result of products of a n-dimensional geometric algebra are always 2^n-dimensional multivectors. Each of the 2^n coefficients can be computed as a sum of (signed or unsigned) products of coefficients of the multivectors to be multiplied (as indicated in Figure 5). We distribute this computation to 2^n threads, each computing one coefficient.

```
Calculate(const float* A, const float* B, float* C) {
    int coeff = threadIdx.x;
    float res = 0.0;
    int offset = ...;
    int length = ...;
    for each relevant summand:
    {
      Summand s = ...;
      res += A[s.first] * B[s.second] * s.sign;
    }
    C[coeff] = res;
}
```

Figure 6: Pseudo code for the computation of one coefficient of a geometric product.

Figure 6 describes the specific kernel code for one thread, each computing one coefficient of the 2^n-dimensional multivector.

Please find some details on this application in [27].

5 RESULTS

Our geometric algebra computing approach is able to generate implementations for different sequential and parallel platforms (see Figure 3). While some of the described implementations are still work in progress, we already have results for implementations in C, for a FPGA and for CUDA.

Our first test case was the inverse kinematics of the arm of a virtual character in a virtual reality application. Naively implemented on a sequential processor platform, the first geometric algebra algorithm was initially slower than the conventional one. However, with our Maple based optimization approach the software implementation became three times faster [14] than the conventional solution. The hardware implementation on a FPGA (as described in section 4.1) became even 300 times faster [15].

The results of our CUDA implementation of arbitrary geometric products can be found in [27].

Recently we investigated the runtime performance of a robotics grasping algorithm described in geometric algebra [28]. It turned out that the implementation on a sequential processor was 14 times faster and on the CUDA platform 44 times faster than the solution with conventional mathematics.

6 CONCLUSION AND FUTURE WORK

We presented the currently most suitable geometric algebra computing platforms. For the adaptation of the algorithms to the different platforms we presented our compilation approach. While the Maple based compilation approach is able to handle algebras up to a dimension of 9, the table based approach is restricted by the memory needed for the size of the multiplication tables. These tables are exponentially increasing with the dimension of the algebra. In this context, investigations for lower amounts of memory are needed, for instance the implementation on a multiplicative basis as described in [5].

Currently, the presented parallel computing platforms can be seen as approximations to perfect GA computers. As a long-term vision, we hope that this research will lead to computing platforms optimally supporting GA computers in the future.

ACKNOWLEDGEMENTS

This work was supported by the DFG (Deutsche Forschungsgemeinschaft) project HI 1440/1-1.

A MULTIPLICATION TABLES

Table 1: Multiplication table describing the geometric product of two multivectors $a = \sum a_i E_i$ and $b = \sum b_i E_i$ for the 3D euclidean GA. Each entry describes the geometric product of two basis blades E_i and E_j expressed in terms of the basis blades E_k. Each coefficient c_k of the product $c = ab$ can be computed by summing up the products $\pm a_i * b_j$ based on the E_k table entries, for instance $c_1 = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + a_4 * b_4 - a_5 * b_5 - a_6 * b_6 - a_7 * b_7 - a_8 * b_8$ for the E_1 table entries

	b	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a		1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
E_1	1	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
E_2	e_1	E_2	E_1	E_5	E_7	E_3	E_8	E_4	E_6
E_3	e_2	E_3	$-E_5$	E_1	E_6	$-E_2$	E_4	$-E_8$	$-E_7$
E_4	e_3	E_4	$-E_7$	$-E_6$	E_1	E_8	$-E_3$	$-E_2$	E_5
E_5	e_{12}	E_5	$-E_3$	E_2	E_8	$-E_1$	E_7	$-E_6$	$-E_4$
E_6	e_{23}	E_6	E_8	$-E_4$	E_3	$-E_7$	$-E_1$	E_5	$-E_2$
E_7	e_{13}	E_7	$-E_4$	$-E_8$	E_2	E_6	$-E_5$	$-E_1$	E_3
E_8	e_{123}	E_8	E_6	$-E_7$	E_5	$-E_4$	$-E_2$	E_3	$-E_1$

Table 2: Multiplication table describing the outer product of two general multivectors $a = \sum a_i E_i$ and $b = \sum b_i E_i$ for the 3D euclidean GA.

	b	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a		1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
E_1	1	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
E_2	e_1	E_2	0	E_5	E_7	0	E_8	0	0
E_3	e_2	E_3	$-E_5$	0	E_6	0	0	$-E_8$	0
E_4	e_3	E_4	$-E_7$	$-E_6$	0	E_8	0	0	0
E_5	e_{12}	E_5	0	0	E_8	0	0	0	0
E_6	e_{23}	E_6	E_8	0	0	0	0	E_5	0
E_7	e_{13}	E_7	0	$-E_8$	0	0	0	0	0
E_8	e_{123}	E_8	0	0	0	0	0	0	0

Table 3: Multiplication table describing the geometric product of two vectors $a = a_1 e_1 + a_2 e_2 + a_3 e_3$ and $b = b_1 e_1 + b_2 e_2 + b_3 e_3$ for the 3D euclidean GA. Note that all the rows and columns for basis blades not needed for the vectors are set to 0.

		b		**b_1**	**b_2**	**b_3**				
			E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a			1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
	E_1	1	0	0	0	0	0	0	0	0
a_1	E_2	e_1	0	E_1	E_5	E_7	0	0	0	0
a_2	E_3	e_2	0	$-E_5$	E_1	E_6	0	0	0	0
a_3	E_4	e_3	0	$-E_7$	$-E_6$	E_1	0	0	0	0
	E_5	e_{12}	0	0	0	0	0	0	0	0
	E_6	e_{23}	0	0	0	0	0	0	0	0
	E_7	e_{13}	0	0	0	0	0	0	0	0
	E_8	e_{123}	0	0	0	0	0	0	0	0

REFERENCES

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

[2] William Kingdon Clifford. *Applications of Grassmann's Extensive Algebra*, volume 1 of *American Journal of Mathematics*, pages 350–358. The Johns Hopkins University Press, 1878.

[3] Crispin Deul, Michael Burger, Dietmar Hildenbrand, and Andreas Koch. Raytracing point clouds using geometric algebra. In *submitted to the proceedings of the GraVisMa workshop, Plzen*, 2010.

[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. *Efficient Implementation of Geometric Algebra*. PhD thesis, University of Amsterdam, 2007.

[6] 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.

[7] 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.

[8] Hermann Grassmann. *Die Ausdehnungslehre*. Verlag von Th. Chr. Fr. Enslin, Berlin, 1862.

[9] David Hestenes. *Space-Time Algebra (Documents on Modern Physics)*. Gordon and Breach, 1966.

[10] David Hestenes. *New Foundations for Classical Mechanics*. Dordrecht, 1986.

[11] 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.

[12] David Hestenes and Garret Sobczyk. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht, 1984.

[13] David Hestenes and Renatus Ziegler. Projective Geometry with Clifford Algebra. *Acta Applicandae Mathematicae*, 23:25–63, 1991.

[14] 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.

[15] Dietmar Hildenbrand, Holger Lange, Florian Stock, and Andreas Koch. Efficient inverse kinematics algorithm based on conformal geometric algebra using reconfigurable hardware. In *GRAPP conference Madeira*, 2008.

[16] Dietmar Hildenbrand and Joachim Pitt. The Gaalop home page. Available at http://www.gaalop.de, 2008.

[17] Dietmar Hildenbrand, Joachim Pitt, and Andreas Koch. Gaalop - high performance parallel computing based on conformal geometric algebra. In Eduardo Bayro-Corrochano and Gerik Scheuermann, editors, *Geometric Algebra Computing for Engineering and Computer Science*. Springer, 2009.

[18] Intel. Ct: C for throughput computing home page. Available at http://techresearch.intel.com/articles/Tera-Scale/1514.htm, 2009.

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

[20] The homepage of maple. Available at http://www.maplesoft.com/products/maple, 2009. 615 Kumpf Drive, Waterloo, Ontario, Canada N2V 1K8.

[21] Biswajit Mishra and Peter Wilson. Hardware implementation of a geometric algebra processor core. In *Proceedings of IMACS International Conference on Applications of Computer Algebra (in press), Nara, Japan*, 2005.

[22] NVIDIA. The CUDA home page. Available at http://www.nvidia.com/object/cuda_home.html, 2009.

[23] Christian Perwass. The CLU home page. Available at http://www.clucalc.info, 2005.

[24] 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.

[25] Hans-Joachim Petsche. The Grassmann Bicentennial Conference home page. Available at http://www.uni-potsdam.de/u/philosophie/grassmann/Papers.htm, 2009.

[26] Alyn Rockwood and Dietmar Hildenbrand. Engineering graphics in geometric algebra. In Eduardo Bayro-Corrochano and Gerik Scheuermann, editors, *Geometric Algebra Computing for Engineering and Computer Science*. Springer, 2009.

[27] Christian Schwinn, A Goerlitz, and Dietmar Hildenbrand. Geometric algebra computing on the cuda platform. In *submitted to the proceedings of the GraVisMa workshop, Plzen*, 2010.

[28] Florian Wörsdörfer, Bayro-Corrochano Eduardo Stock, Florian, and Dietmar Hildenbrand. Optimization and performance of a robotics grasping algorithm described in geometric algebra. In *Iberoamerican Congress on Pattern Recognition 2009, Guadalajara, Mexico*, 2009.

Exponentials and Motions in Geometric Algebra

Leon Simpson
Department of Mechanical Engineering
University of Bath
Bath BA2 7AY
United Kingdom
l.c.simpson@bath.ac.uk

Glen Mullineux
Department of Mechanical Engineering
University of Bath
Bath BA2 7AY
United Kingdom
g.mullineux@bath.ac.uk

ABSTRACT

The use of geometric algebra to define and manipulate rigid-body motions is investigated. An algebra with four basis elements of grade 1 is used in which the square of one of these elements is regarded as being infinite. This gives a representation of projective space and allows rotations and translations to be defined exactly. By smoothly interpolating between such transforms, smooth motions can be created using techniques such as spherical linear interpolation (Slerp). This requires the ability to handle the exponential function within the algebra. A closed form expression for the exponential is derived in the general case when the square of the special basis element is any real number. Taking this to be infinite allows smooth motions to be created and some examples are presented.

Keywords
Geometric algebra, exponential function, smooth motion, interpolation, Slerp.

1. INTRODUCTION

The ideas of geometric (Clifford) algebras have been known since the 1800's. However they fell into disuse and interest was only regained in the last ten years or so [1]. Now they are used in many areas including quantum physics [2] and computer vision [3]. One particular application area is that of creating smooth motions such as those required for the kinematics of robotic and mechanism systems [4]. For this, geometric algebra can be regarded as a technique related to the use of quaternions and dual quaternions [5, 6].

There is interest in the use of geometric algebra to represent rigid-body transforms (translations and rotations) in the same form. One approach is to do this using the ideas of conformal geometric algebra [7]. An alternative approach is to work with a "conventional" formulation of a geometric algebra and arrange that the square of one of the basis vectors is treated as being infinite [8, 9].

This is done by treating infinity (symbolically) as the reciprocal of a small real scalar ε that is allowed to become vanishing small.

An overview of the algebra $\mathcal{G}_4$ constructed in this way is given in section 2. For the purposes of this paper, the notation used involves a real number λ where $\lambda^2 = \varepsilon^{-1}$ and consequently λ is regarded as being infinite. The approach allows the algebra to form a model of projective geometry and allows exact representations to be obtained for both rotations and translations (section 3).

One of the reasons why quaternions, in particular, are now used regularly for rotations in computer graphics is the ability to interpolate using them in equal steps [10]. This has given rise to the construction known as the *Slerp* which effectively makes use of the exponential function. Exponentials can also be used to similar effect in conformal geometric algebra [11] and with matrix representation of transforms [12].

Quaternions naturally represent rotations, but, in their basic form, they cannot cope with translations. However, the geometric algebra $\mathcal{G}_4$ (which contains the quaternions) can deal with both types of transforms [9]. The aim of this paper is therefore to investigate the evaluation of the exponential of even-grade elements in $\mathcal{G}_4$. This can be done in several ways. One is to use the power series definition of the exponential function and directly substitute the appropriate even-grade element. The expression can be simplified by ignoring terms which involve powers of ε which are two or higher as these are small. Another way is to make use of Chasles's theorem. This says that every rigid-body transform can be expressed as the product of a rotation and a translation that commute. The exponentiation of these individual terms is straightforward and the commutivity allows the product of individual exponentials to be formed to give the exponential of the original transform.

The interest is in an approach which does not rely on the fact that the parameter λ is infinite. Use is made of a representation of the algebra in terms of matrices over the quaternions; this is discussed in section 4. This gives an expression, derived in section 5, for the exponential which is closed in the sense that it does not require explicit use of power series. It is also applicable for any value of λ. This parameter can subsequently be allowed to pass to infinity in order to make use of the transforms that have been constructed.

Finally, in section 6, use is made of the exponential function (in the case when λ is infinite) to obtain smooth motions between prescribed poses (positions) and some examples are given. These are simple examples of the sorts of motions required in a variety of application areas including: robotics [3], mechanism simulation [4, 5], human movement [11], and general three dimensional geometry [8, 13].

2. GEOMETRIC ALGEBRA $\mathcal{G}_4$

The geometric algebra $\mathcal{G}_4$ is constructed as follows. Start with a four-dimensional vector space defined over the real numbers $\mathbb{R}$ whose basis vectors (the underlying basis) are denoted by e_0, e_1, e_2, e_3. This is extended to a vector space of dimension 16, whose basis elements are denoted by e_σ where σ is a subset of $\{0, 1, 2, 3\}$.

A multiplication between basis elements is now available by using the idea that if $\sigma = \{a, b, \ldots, d\}$, then $e_a e_b \ldots e_d = e_\sigma$. So, for example

$$e_1 e_2 = e_{12} \qquad \text{and} \qquad e_1 e_2 e_3 = e_{123}$$

The multiplication is not commutative since the underlying basis vectors are defined to anti-commute. For example,

$$e_{12} = e_1 e_2 = -e_2 e_1 = -e_{21}$$

To simplify more complicated products, it is necessary to specify what is the square of each original basis element. These are taken as real numbers and it follows that all basis elements also square to real numbers. The usual practice is to define the squares e_i^2 to be 0 or ± 1. However, here the following squares are used.

$$e_0^2 = \varepsilon^{-1} \qquad\qquad e_1^2 = e_2^2 = e_3^2 = 1$$

where ε is a positive real number. As discussed later, there is interest in the case when ε approaches zero. For convenience, the positive real number λ is defined so that $\lambda^2 = \varepsilon^{-1}$. When the limit is taken, λ tends to infinity.

Given these squares, more general products of the basis elements can be simplified as the following example shows.

$$\begin{aligned} e_0 e_{012} e_{13} &= e_0 e_0 e_1 e_2 e_1 e_3 \\ &= -e_0 e_0 e_1 e_1 e_2 e_3 \\ &= -\varepsilon^{-1} e_{23} \\ &= -\lambda^2 e_{23} \end{aligned}$$

The basis element e_ϕ, where ϕ is the empty set, acts as the real number 1 and is identified with it. The basis element e_{0123} anti-commutes with all the original basis vectors. It is here denoted by ω.

The general element of $\mathcal{G}_4$ is a linear combination of the basis elements

$$x = \sum_\sigma C_\sigma e_\sigma \qquad (1)$$

and addition and multiplication extend as with any other space.

The *grade* of a basis element e_σ is the size of the subset σ. If the only basis elements involved (with non-zero coefficients) in a general element x have the same grade, then this is taken as the grade of the element x. Such elements are also referred to as *blades* or i-blades, where i is the grade in question.

Elements of grade 1 are called *vectors*, those of grade 2 are *bivectors*, and those of grade 3 are *trivectors*. The basis element ω is called the *pseudo-scalar*, and scalar multiples of ω are the only elements of grade 4 in the algebra.

If λ is taken as zero (rather than being large), so that $e_0^2 = 0$, then $\mathcal{G}_4$ provides a representation of projective space with vectors corresponding to planes, bivectors to lines, and trivectors to points [13]. Conversely, when $\lambda^2 = \varepsilon^{-1}$ where ε is (vanishingly) small, the algebra gives an alternative representation in which vectors correspond to points, bivectors to lines, and trivectors to planes [8].

To be able to deal with the algebra $\mathcal{G}_4$ computationally, it is necessary to be able to handle the ε quantity. This has been achieved by means of a suite of procedures written in C++. Effectively, these treat the scalars not simply as real numbers but as real polynomials in ε in which both positive and negative powers are allowed to appear. A C++ class allows such polynomials to be created and handles arithmetical operations involving them.

These procedures allow geometric calculations to be be performed. When the results are required for display purposes, these (usually) involve points and the result of the appropriate evaluation takes the form

$$\alpha(e_0 + Xe_1 + Ye_2 + Ze_3) + O(\varepsilon)$$

where α is a non-zero scalar. At this stage, the terms involving ε are ignored which is equivalent to letting ε tend to zero, and the result then represents the cartesian point (X, Y, Z).

3. TRANSFORMS

The *reverse* of a basis element e_σ is obtained by reversing the order of the subset σ. The results is either the original element or the negative of it. The reverse of an element is here denoted by a dagger (†) as in the following examples.

$$\begin{aligned} e_{12}^\dagger &= e_{21} = -e_{12} \\ e_{123}^\dagger &= e_{321} = -e_{123} \\ e_{0123}^\dagger &= e_{3210} = e_{0123} \end{aligned}$$

Suppose S is an element of $\mathcal{G}_4$. It defines a map

$$x \mapsto S^\dagger x S$$

on the general element $x \in \mathcal{G}_4$.

Consider the case in which S has even grade. If x is a vector in $\mathcal{G}_4$, then its image also has odd grade and is equal to its own reverse; hence it is also a vector. So the map sends points in the corresponding projective space to other points. It can be verified that this map represents a rigid-body transform.

Furthermore, taking $\lambda^2 = \varepsilon^{-1}$ with ε (vanishingly) small, any combination of rotations and translations can be created as a map in this way [9]. For example, taking

$$S = \cos(\theta) + B\sin(\theta) \tag{2}$$

where B is a unit bivector, the map represents a rotation through an angle 2θ about an axis determined by the line Be_{123}. And when

$$S = 1 + \varepsilon e_0 v \tag{3}$$

where v is a vector not involving e_0, the map creates a translation through $2v$. This can be checked as follows. If $u = Xe_1 + Ye_2 + Ze_3$ so that $x = e_0 + u$ represents the typical point in projective space, then

$$\begin{aligned} S^\dagger x S &= (1 - \varepsilon e_0 v)(e_0 + u)(1 + \varepsilon e_0 v) \\ &= (1 - \varepsilon vv - \varepsilon uv - \varepsilon vu)e_0 \\ &\qquad + u + 2v - \varepsilon vuv \\ &\rightarrow e_0 + u + 2v \end{aligned}$$

where the limit is taken as ε tends to zero.

The even-grade element $(\alpha + \varepsilon\beta\omega)$ acts on vectors to produce the identity transform. This is because, if $u = Xe_1 + Ye_2 + Ze_3$ then

$$(\alpha + \varepsilon\beta\omega)(e_0 + u)(\alpha + \varepsilon\beta\omega) = \alpha^2(e_0 + u)$$

in the limit as ε tends to zero. The real factor α^2 does not affect the point represented in the projective space.

If S is a general even-grade element, then $S^\dagger S = SS^\dagger$ and this is an element of the form $p + \varepsilon q\omega$ where p and q are scalars. Then the even-grade element

$$S^* = \frac{1}{\sqrt{p}}\left[1 - \frac{\varepsilon q\omega}{2p}\right] S$$

is equivalent to S in the sense that it creates the same transform. Furthermore,

$$S^{*\dagger} S^* = 1 = S^* S^{*\dagger}$$

Thus S^* is a normalised form of S. For simplicity, it is now assumed that all even-grade elements used to create transforms have been normalised. This is already true of the elements used to generate the examples of a rotation and a translation in equations (2) and (3).

Suppose an object is defined by points in its own local coordinate system. Then an even-grade element S defines a map into world space which creates a *pose* for that object. Two such elements S_0 and S_1 define two different poses. If one can smoothly interpolate between S_0 and S_1, then a smooth motion from one pose to the other can be created.

One such interpolation can be achieved using the Bézier formulation. This uses the following combination

$$S(t) = S_0(1 - t) + S_1 t$$

where the parameter t varies between 0 and 1.

Another interpolation technique is *Slerp*, spherical linear interpolation, which was originally created for use with the quaternions [10]. This uses the following interpolation formula.

$$S(t) \;=\; S_0\,(S_0^\dagger S_1)^t$$

where again t lies between 0 and 1. At the end points of the motion,

$$\begin{aligned} S(0) &= S_0 \\ S(1) &= (S_0 S_0^\dagger) S_1 \;=\; S_1 \end{aligned}$$

using the assumption that S_0 is normalised, so that motion does pass between the two poses.

If there is an element U such that

$$S_0^\dagger S_1 \;=\; \exp(U) \tag{4}$$

then the interpolation takes the form

$$S(t) \;=\; S_0\,\exp(tU) \tag{5}$$

The next two sections investigate how the exponential function can be evaluated in $\mathcal{G}_4$. This is done in the case in which λ is taken as a general real number with no assumption being made that it is infinite (or even large). Some interesting symmetry and structure is apparent in the results obtained. Naturally when the exponential is used in the context of defining motions, there is a need to let λ tend to infinity. For the examples presented in section 6, this is achieved via the C++ procedures which treat the coefficients as polynomials in ε.

4. MATRIX REPRESENTATION

A representation of the even-grade algebra of $\mathcal{G}_4$ is the set of 2×2 diagonal matrices over the division ring of quaternions. If i, j, k are the standard quaternions, then one representation is the following. (Other variations are possible by changing the signs of i, j, k throughout.)

$$\begin{aligned}
1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
e_{23} &= \begin{bmatrix} -i & 0 \\ 0 & -i \end{bmatrix} \\
e_{13} &= \begin{bmatrix} j & 0 \\ 0 & j \end{bmatrix} \\
e_{12} &= \begin{bmatrix} -k & 0 \\ 0 & -k \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
e_{0123} \;=\; \omega &= \lambda \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
e_{01} &= \lambda \begin{bmatrix} i & 0 \\ 0 & -i \end{bmatrix} \\
e_{02} &= \lambda \begin{bmatrix} j & 0 \\ 0 & -j \end{bmatrix} \\
e_{03} &= \lambda \begin{bmatrix} k & 0 \\ 0 & -k \end{bmatrix}
\end{aligned}$$

This representation leads to an isomorphism between the sub-algebra of even-grade elements of $\mathcal{G}_4$ and the set of diagonal, quaternionic 2×2 matrices.

5. EXPONENTIATION

The aim here is to find an explicit form for $\exp(x)$ where x is an even-grade element of $\mathcal{G}_4$. If such an expression can be found, then it avoids the need to evaluate the exponential in terms of its power series. Following equation (1), the typical even-grade element has the form

$$x \;=\; C_\phi \;+\; B \;+\; C_\omega \omega$$

where B is the bivector

$$\begin{aligned} B \;=\; & C_{01}e_{01} \;+\; C_{02}e_{02} \;+\; C_{03}e_{03} \\ & + C_{12}e_{12} \;+\; C_{13}e_{13} \;+\; C_{23}e_{23} \end{aligned}$$

Since ω and, of course, 1 commute with all even-grade elements, it follows that

$$\exp x \;=\; \exp(C_\phi)\,\exp(C_\omega \omega)\,\exp(B)$$

The first factor here is a real number. The second factor can be dealt with by noting that $\omega^2 = \lambda^2$.

$$\begin{aligned}
& \exp(C_\omega \omega) \\
&= 1 + C_\omega \omega + \frac{C_\omega^2 \omega^2}{2!} + \frac{C_\omega^3 \omega^3}{3!} \\
&\quad + \frac{C_\omega^4 \omega^4}{4!} + \frac{C_\omega^5 \omega^5}{5!} + \ldots \\
&= \left[1 + \frac{C_\omega^2 \omega^2}{2!} + \frac{C_\omega^4 \omega^4}{4!} + \ldots \right] \\
&\quad + \left[C_\omega \lambda + \frac{C_\omega^3 \lambda^3}{3!} + \frac{C_\omega^5 \lambda^5}{5!} + \ldots \right] \frac{\omega}{\lambda} \\
&= \cosh(C_\omega \lambda) \;+\; \frac{\sinh(C_\omega \lambda)}{\lambda}\,\omega
\end{aligned} \tag{6}$$

Before considering $\exp(B)$, note that if

$$q = q_1 i + q_2 j + q_3 k$$

is a quaternion, then

$$q^2 = -(q_1^2 + q_2^2 + q_3^2) = -\gamma^2 \quad \text{say.}$$

Set $\hat{q} = q/\gamma$ which is the corresponding unit quaternion. Then $\exp(q)$ can be found, in a similar fashion to $\exp(C_\omega \omega)$, as follows.

$$\begin{aligned} & \exp(q) \\ = \; & 1 + q + \frac{q^2}{2!} + \frac{q^3}{3!} + \frac{q^4}{4!} + \frac{q^5}{5!} + \dots \\ = \; & 1 + q - \frac{\gamma^2}{2!} - \frac{\gamma^3 q}{3!} + \frac{\gamma^4}{4!} + \frac{\gamma^5 q}{5!} - \dots \\ = \; & (\cos\gamma) \; + \; (\sin\gamma)\hat{q} \end{aligned}$$

This, of course, is equivalent to Euler's formula for complex numbers.

Using the results of the last section, the 2×2 matrix corresponding to the bivector B is the following.

$$B \;=\; \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

where

$$\begin{aligned} a &= a_1 i + a_2 j + a_3 k \\ b &= -b_1 i - b_2 j - b_3 k \end{aligned}$$

and

$$\begin{aligned} a_1 &= (\lambda C_{01} - C_{23}) \\ a_2 &= (\lambda C_{02} + C_{13}) \\ a_3 &= (\lambda C_{03} - C_{12}) \\[1em] b_1 &= (\lambda C_{01} + C_{23}) \\ b_2 &= (\lambda C_{02} - C_{13}) \\ b_3 &= (\lambda C_{03} + C_{12}) \end{aligned}$$

Hence $\exp(B)$ is the matrix

$$\begin{bmatrix} (\cos\alpha) + (\sin\alpha)\hat{a} & 0 \\ 0 & (\cos\beta) + (\sin\beta)\hat{b} \end{bmatrix}$$

where

$$\begin{aligned} \alpha &= \sqrt{[a_1^2 \; + \; a_2^2 \; + \; a_3^2]} \\ \beta &= \sqrt{[b_1^2 \; + \; b_2^2 \; + \; b_3^2]} \end{aligned}$$

The matrix for $\exp(B)$ is now considered on a term-by-term basis. Firstly the scalar part is given as follows.

$$\begin{aligned} & \begin{bmatrix} \cos\alpha & 0 \\ 0 & \cos\beta \end{bmatrix} \\ = \; & \tfrac{1}{2}(\cos\alpha + \cos\beta) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ & + \tfrac{1}{2}(\cos\alpha - \cos\beta) \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ = \; & \tfrac{1}{2}[\cos\alpha + \cos\beta] + \tfrac{1}{2}[\cos\alpha - \cos\beta]\frac{\omega}{\lambda} \end{aligned}$$

The term involving i is next considered.

$$\begin{aligned} & \begin{bmatrix} \frac{\sin\alpha}{\alpha} a_1 & 0 \\ 0 & -\frac{\sin\beta}{\beta} b_1 \end{bmatrix} i \\ = \; & \tfrac{1}{2}\left(\tfrac{\sin\alpha}{\alpha} a_1 - \tfrac{\sin\beta}{\beta} b_1\right) \begin{bmatrix} i & 0 \\ 0 & i \end{bmatrix} \\ & + \tfrac{1}{2}\left(\tfrac{\sin\alpha}{\alpha} a_1 + \tfrac{\sin\beta}{\beta} b_1\right) \begin{bmatrix} i & 0 \\ 0 & -i \end{bmatrix} \\ = \; & -\tfrac{1}{2}\left(\tfrac{\sin\alpha}{\alpha} a_1 - \tfrac{\sin\beta}{\beta} b_1\right) e_{23} \\ & + \tfrac{1}{2}\left(\tfrac{\sin\alpha}{\alpha} a_1 + \tfrac{\sin\beta}{\beta} b_1\right) \frac{e_{01}}{\lambda} \end{aligned}$$

The terms involving j and k can be dealt with similarly. Combining the results yields the required expression for $\exp(B)$.

$$\begin{aligned} \exp(B) \;=\; & \tfrac{1}{2}[\cos\alpha + \cos\beta] \\ & + \tfrac{1}{2\lambda}\left[\tfrac{\sin\alpha}{\alpha}a_1 + \tfrac{\sin\beta}{\beta}b_1\right] e_{01} \\ & + \tfrac{1}{2\lambda}\left[\tfrac{\sin\alpha}{\alpha}a_2 + \tfrac{\sin\beta}{\beta}b_2\right] e_{02} \\ & + \tfrac{1}{2\lambda}\left[\tfrac{\sin\alpha}{\alpha}a_3 + \tfrac{\sin\beta}{\beta}b_3\right] e_{03} \\ & - \tfrac{1}{2}\left[\tfrac{\sin\alpha}{\alpha}a_3 - \tfrac{\sin\beta}{\beta}b_3\right] e_{12} \\ & + \tfrac{1}{2}\left[\tfrac{\sin\alpha}{\alpha}a_2 - \tfrac{\sin\beta}{\beta}b_2\right] e_{13} \\ & - \tfrac{1}{2}\left[\tfrac{\sin\alpha}{\alpha}a_1 - \tfrac{\sin\beta}{\beta}b_1\right] e_{23} \\ & + \tfrac{1}{2\lambda}[\cos\alpha - \cos\beta]\omega \end{aligned} \tag{7}$$

Some examples are now given.

The first example is to find $\exp(\varepsilon d e_0 u)$ where $\varepsilon = \lambda^{-2}$ is vanishingly small, d is a scalar, and $u = u_1 e_1 + u_2 e_2 + u_3 e_3$ is a vector of unit length. This means that, in the previous notation, C_ϕ and C_ω are both zero, and the only non-zero coefficients within the bivector part are $C_{01} = \varepsilon du_1$, $C_{02} = \varepsilon du_2$, $C_{03} = \varepsilon du_3$. Hence

$$\alpha = \beta = \lambda^{-1} d \sqrt{[u_1^2 + u_2^2 + u_3^2]} = \lambda^{-1} d$$

which is small, so that its cosine is unity, and $(\sin\alpha)/\alpha = (\sin\beta)/\beta = 1$. Using equation (7) gives

$$\begin{aligned} & \exp(\varepsilon d e_0 u) \\ = \; & 1 + C_{01}e_{01} + C_{02}e_{02} + C_{03}e_{03} \\ = \; & 1 + \varepsilon d e_0 u \qquad (8) \end{aligned}$$

which, if $v = du$, is the same as the element creating a translation in equation (3).

The second example is to find $\exp(B\theta)$ in the case when $B = C_{12}e_{12} + C_{13}e_{13} + C_{23}e_{23}$ is a unit bivector not involving e_0, so that $C_{12}^2 + C_{13}^2 + C_{23}^2 = 1$. As before, C_ϕ and C_ω are both zero and $\exp(C_\phi) = \exp(C_\omega) = 1$. Additionally, $\alpha = \beta = \theta$. So, from equation (7), it is seen that

$$\begin{aligned} & \exp(B\theta) \\ = \; & \cos\theta + \tfrac{\sin\theta}{\theta}[C_{12}e_{12} + C_{13}e_{13} + C_{23}e_{23}]\theta \\ = \; & \cos\theta + (\sin\theta)B \qquad (9) \end{aligned}$$

which creates a rotation as in equation (2).

6. EXAMPLES

Some examples based around the idea of interpolating between two poses are presented. In each case an L-shaped block is the object that is moved. This is defined by its vertices (and their interconnections) which have coordinates specified in a local frame of reference. Two even-grade elements S_0 and S_1 are used to map from the local frame into world space thus defining two poses for the block. The interest is in smoothly interpolating a motion between them. Interpolation based around the idea of the Slerp, equation (5), is used. This creates an even-grade element $S(t)$ for $0 \le t \le 1$ with $S(0) = S_0$ and $S(1) = S_1$.

In the first example, the interpolation is pure translation. The initial pose is created at the origin so that

$$S_0 = 1$$

The second pose is a translation of 8 units in the x-direction and 6 units in the y-direction. The corresponding even-grade element is

$$S_1 = 1 + 4\varepsilon e_{01} + 3\varepsilon e_{02}$$

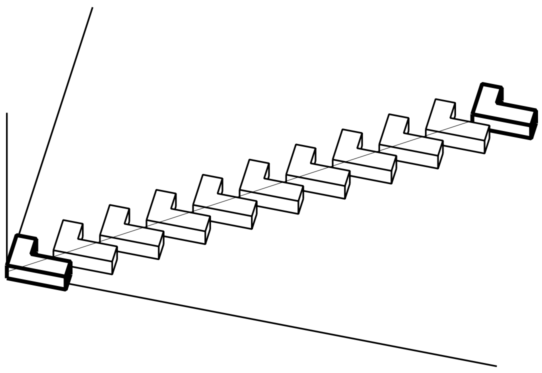


Figure 1: Pure translational motion

as in equation (3). Also needed is the element U such that equation (4) is valid. This leads to

$$U = 4\varepsilon e_{01} + 3\varepsilon e_{02}$$

as in equation (8).

These even-grade elements form the ingredients for the associated Slerp, $S(t)$. This is evaluated for equally spaced values of the parameter t and each is used to transform the block. The results are shown in figure 1. The block translates between the initial and final poses in equal steps.

The second example leads to an interpolation which represents a pure rotation. The first pose arises from a translation of 8 units in the x-direction and is obtained with the following even-grade element.

$$S_0 = 1 + 4\varepsilon e_{01}$$

The second pose is obtained by initially rotating through a right angle about the z-axis and then translating 8 units in the y-direction. Combining the corresponding even-grade elements gives

$$S_1 = \tfrac{1}{\sqrt{2}}\,[1 + 4\varepsilon e_{01} + 4\varepsilon e_{02} + e_{12}]$$

This means that

$$S_0^\dagger S_1 = \tfrac{1}{\sqrt{2}}\,[1 + e_{12}]$$

and equation (9) shows that appropriate value for element U is the following.

$$U = \tfrac{\pi}{4}\, e_{12}$$

Figure 2 shows the resultant interpolation, again evaluated for equally spaced values of the parameter t.

The next example is a variation on the last. The initial pose is obtained as before, but the final pose now involves a rotation through half a revolution

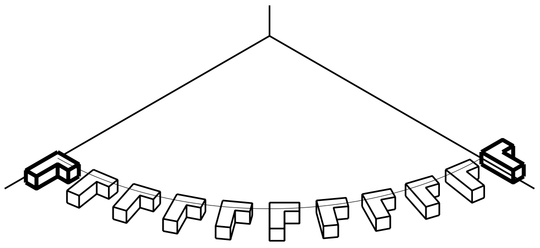


Figure 2: Pure rotational motion

followed by a translation of 8 units in the negative x-direction and a translation of 8 units in the z-direction. This leads to

$$S_1 = 4\varepsilon e_{02} + e_{12} + 4\varepsilon\omega$$

which means that

$$S_0^\dagger S_1 = e_{12} + 4\varepsilon\omega$$

and the required element U is then the following.

$$U = \tfrac{\pi}{2}\, e_{12} \;+\; 4\varepsilon e_{03}$$

This can be checked using equation (7). In the previous notation all the C_{ij} are zero except for $C_{12} = \pi/2$ and $C_{03} = 4\varepsilon = 4\lambda^{-2}$. Hence

$$\begin{aligned} a_1 = a_2 = 0 = b_1 = b_2 \\ \alpha = a_3 = 4\lambda^{-1} - \pi/2 \\ \beta = b_3 = 4\lambda^{-1} + \pi/2 \\ \sin\alpha = -1 = -\sin\beta \\ \cos\alpha = -4\lambda^{-1} = -\cos\beta \end{aligned}$$

Figure 3 shows the motion with the block following a helical path. Figure 5 shows the same motion as viewed in the z-direction and this shows that the path is a circular helix.

Note that U is the sum of two even-grade elements which commute. This means that $\exp U$ is a product of the exponentials of the two terms. Set $R = \exp(\frac{\pi}{2}e_{12})$ and $T = \exp(4\varepsilon e_{03})$, and then

$$S_0^\dagger S_1 = \exp U = RT$$

Using equations (8) and (9) it is seen that R is a rotation about the z-axis and T is a translation in the z-direction. This is a case of Chasles's theorem where the motion is expressed a combination of a rotation about an axis and a translation along it.

The last example generates the same helical interpolation but now has the block rotating about the direction of motion. To achieve this, the Slerp formulation, equation (5), is modified to the following.

$$S(t) = R^t\, S_0\, (S_0^\dagger S_1)^t = \exp(tr)\, S_0\, \exp(tU)$$

where $R = \exp(r)$ is an even-grade element which generates a rotation. Expressing R as an exponential allows powers of it to be evaluated. Here the following choices are made

$$\begin{aligned} r &= \tfrac{\pi}{2}\, e_{13} \\ R &= e_{13} \end{aligned}$$

So R is a rotation about the y-axis. The resultant motion is shown in figure 5.

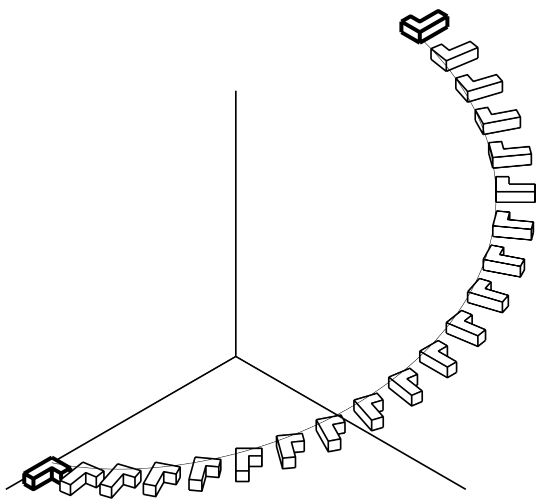


Figure 3: Helical motion which a combined rotation and translation

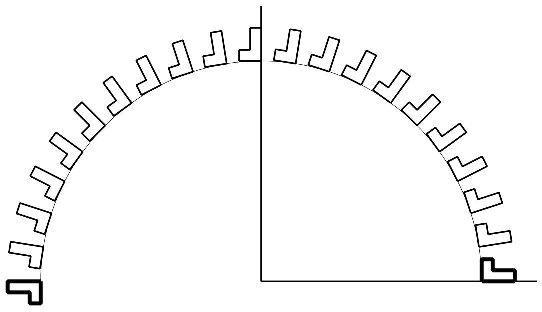


Figure 4: View of helical motion along the axis

7. CONCLUSIONS

The geometric algebra $\mathcal{G}_4$ has been constructed with the square of one of its basis vectors, e_0, being equal to the real number λ^2. If this is regarded as being infinite, the algebra represents projective space and even-grade elements can be used to create exact rigid-body transforms (combinations of translations and rotations).

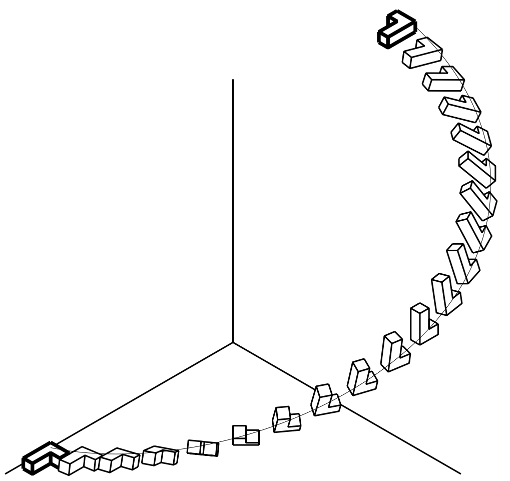


Figure 5: Helical interpolation with spin about the direction of motion

Such transforms allow poses of any object to be created in world space. Smooth motions can be generated which interpolate between pairs of such poses. One interpolation formulation is the Slerp operation. This requires the evaluation of the exponential of even-grade elements of $\mathcal{G}_4$.

A closed form for such exponentials has been obtained which does not rely (explicitly) upon the evaluation of a power series. This has been obtained by using a representation of the even-grade elements in terms of 2×2 diagonal matrices over the quaternions. The expression for the exponential, with λ being regarded as infinite, has been used to obtain examples of smooth motions.

8. ACKNOWLEDGMENTS

The first author is supported by a research studentship provided by the Innovative Design and Manufacturing Research Centre at the University of Bath which is funded by the Engineering and Physical Sciences Research Council (EPSRC). This support is gratefully acknowledged.

References

[1] Lee, C.-C., Stammers, C. W. and Mullineux, G. On the historical overview of geometric algebra for kinematics of mechanisms, in International Symposium on History of Machines and Mechanisms. Yan, H.-S. and Ceccarelli, M., eds., Springer, Berlin, pp. 21-34, 2009.

[2] Parker, R. and Doran, C. Analysis of one and two particle quantum systems using geometric algebra, in Applications of Geometric Algebra in Computer Science and Engineering. Dorst, L., Doran, C. and Lasenby, J., eds., Birkhäuser, Boston, pp. 213-226, 2002.

[3] Bayro-Corrochano, E., Reyes-Lozano, L. and Zamora-Esquivel, J. Conformal geometric algebra for robotic vision. Journal of Mathematical Imaging and Vision, 24, pp. 55-81, 2006.

[4] Etzel, K. R. and McCarthy, J. M. Interpolation of spatial displacements using the Clifford algebra of E^4. Journal of Mechanical Design, 121, pp. 39-44, 1999.

[5] Purwar, A., Jin, Z. and Ge, Q. J. Rational motion interpolation under kinematic constraints of spherical 6R closed chains. Journal of Mechanical Design, 130, pp. 062301:1-9, 2008.

[6] Fang, Y. C., Hsieh, C. C., Kim, M. J., Chang, J. J. and Woo, T. C. Real time motion fairing with unit quaternions. Computer-Aided Design, 30, pp. 191-198, 1998.

[7] Lasenby, A., Doran, C. and Lasenby, R. Rigid body dynamics and conformal geometric algebra, in Proc. Computer Graphics, Vision and Mathematics (GraVisMa) 2009. Hildenbrand, D. and Skala, V., eds., University of West Bohemia, Plzen, September 2009.

[8] Mullineux, G. Clifford algebra of three dimensional geometry. Robotica, 20, pp. 687-697, 2002.

[9] Mullineux, G., Modelling spatial displacements using Clifford algebra. Journal of Mechanical Design, 126, pp. 420-424, 2004.

[10] Shoemake, K. Animating rotation with quaternion curves. ACM Siggraph, 19, pp. 245-254, 1985.

[11] Wareham, R. and Lasenby, J. Mesh vertex pose and position interpolation using geometric algebra, in Articulated Motion and Deformable Objects, 5th International Conference, AMDO 2008. Perales, F. J. and Fisher, R. B. (eds.), Springer, Berlin, pp. 122-131, 2008.

[12] Özgören, M. K. Kinematics analysis of spatial mechanical systems using exponential rotation matrices. Journal of Mechanical Design, 129, pp. 1144-1152, 2007.

[13] Selig, J. M. Clifford algebra of points, lines and planes. Robotica, 20, pp. 545-556, 2000.

Pose estimation based on Geometric Algebra

Yan Cui
DFKI
Trippstadter Strasse. 122
67663 Kaiserslautern Germany
Yan.Cui@dfki.de

Dietmar Hildenbrand
TU Darmstadt
Hochschulstrasse. 10
64283 Darmstadt Germany
Hildenbrand@gris.informatik.tu-darmstadt.de

ABSTRACT

2D-3D pose estimation is an important task for computer vision, ranging from robot navigation to medical intervention. In such applications as robot guidance, the estimation procedure should be fast and automatic, but in industrial metrology applications, the precision is typically a more important factor. In this paper, a new 3D approach for infrared data visualization precisely with the help of 2D-3D pose estimation based on Geometric Algebra is proposed. The approach provides a user friendly interface, a flexible structure and a precise result, which can be adjusted to almost all the geometrically complex objects.

Keywords: Geometric algebra, 2D-3D pose estimation, ICP algorithm.

1 INTRODUCTION

2D-3D pose estimation is an important problem in computer vision. The standard requisites to the pose estimation procedures are high speed, automatic mode and high precision. The main aim in these procedures is to define the relative position and orientation of a known 3D object with respect to a reference camera system. In other words, we search for a transformation (i.e. the pose) of the 3D object such that the transformed object corresponds to 2D image data. For rigid objects, this transformation should be the Euclidean transformation consisting of a rotation R and a translation t. Pose estimation is a subclass of the more general registration problem. The main focus in this paper is given to the pose estimation based on Geometric Algebra and the 3D data visualization with texture mapping. This leads to three main questions:

- How and what kind of image and object features to extract?
- How to do the pose estimation precisely and fast?
- How to detect object parts (surfaces) are visible?

Note that throughout this paper the 3D object model (independent of its representation) is assumed to be known (3D object model is given .wrl file format). The problem how the model of unknown object can be obtained is discussed in works by N. Krueger [21] and M. Zerroug [32].

A 3D object can contain different features like 3D points, 3D lines, 3D spheres, 3D circles, kinematic chain segments, boundary contours and contour parts. The aim is to find the rotation R and the translation t of the object which leads to the best fit of the reference model with the actually extracted entities. So far, it is not defined how to measure the fit quality. It is clear by intuition that a mathematical formalization is not trivial.Current approaches to pose estimation (and registration in general) can be divided into two categories:

- Explicit pose estimation [28]: The involved 2D and 3D entities are defined explicitly. This includes points, lines and higher order entities such as conics, kinematics chains or higher order 3D curves.
- Free-form pose estimation [28]: The involved entities are modeled as free-form objects such as parametric curves/surfaces, 3D meshes, active contours and implicit curves/surfaces.

Additionally, from a statistical point of view, pose estimations of global object descriptions are more accurate and robust than those from a sparse set of local features. But on the other hand, pose estimation based feature can be performed much faster. In this paper we discuss 2D-3D pose estimation using the feature-based method in explicit point corresponding and the free-from method in active contour. After finding the right posed position of the 3D object, we try to visualize the 3D data with texture mapping from the 2D image to the 3D mode, test whether the triangles of the 3D object are visible or not with a ray-tracing algorithm. In this paper we implement the ray-tracing method based on the Geometric Algebra approach in [13].

Main contribution in this work can be generalized as follows:

- We do the camera calibration based on the linear method. This model is used in the geometric algebra framework. The conformal geometric algebra [23] allows to deal with higher order entities (lines, planes, circles, spheres) in the same manner as with points. It is further possible to model the conformal group on these entities by applying special operators in a multiplicative manner.

- This paper introduces a new pose estimation method based on the active object contour extraction. To estimate the pose of free-form contours, ICP (Iterative Closest Point) algorithms [30, 15] are applied. Normal ICP starts with two data sets and an initial guess for their rigid body motion. Then the transformation is refined by repeatedly generating pairs of corresponding point sets and minimizing the error metric. Furthermore, they will later be used to compare a 3D contour, modeled by Fourier descriptors, with 3D reconstructed projection rays. The use of Fourier descriptors is accompanied by some features, which can advantageously be applied within the pose estimation problem: instead of estimating the pose for a whole 3D contour, low-pass descriptions of the contour can be used for an approximation. This leads to a speed up of the algorithm. Meanwhile, this paper brings forward an improved ICP, which improves the normal ICP algorithm to avoid the local minimum.

The paper is structured as follows. In section 2, related work of pose estimation based on the geometric algebra is presented. The 2D-3D entities constraint equations and some experiments of 2D-3D point to line constraints will be given in section 3. Section 4 describes 2D-3D pose estimation based on an active contour method.

2 RELATED WORK

The first pose estimation algorithms were based on a point-based method, which is widely discussed in many foundational papers. A rigid body is generally assumed, but no complete explicit geometric model is given. Methods of this class were firstly studied in the 80's and 90's and pioneering works were done by Lowe [11, 12] and Grimson [22]. Lowe applied a Newton-Raphson minimization method to the pose estimation problem and showed the direct application of numerical optimization techniques in the context of noisy data and in gaining fast (real-time capable) algorithms. Lowe's work is based on pure point concepts and he expresses the constraint equations in the 2D image plane. To linearize the equations, an affine camera model is assumed. The extension to a fully projective formulation is proposed by Araujo et al [1]. The minimum number of correspondences that produce an unique solution are three (non collinear and non-coplanar) points. Four coplanar and non-collinear points also give a unique solution [17]. In general the accuracy increases with the number of used point features. Over-determined solutions are also used for camera calibration [25].

A pose estimation algorithm based on dual quaternions [31] is given by Walker et al. [24]. The method uses the real-part of the dual quaternion to estimate the rotational part and the dual-part of the dual-quaternion to estimate the translational part of the pose. This approach is also discussed by Daniilidis [10] in the context of hand-eye calibration.

There exist some methods that do the pose estimation with image silhouettes (also called occluding contours, extremal contours, apparent contours), which are a rich source of geometric information about the 3D objects. An image silhouette is the projection of the locus of points on the object.

Reconstructing the shape from silhouettes was introduced by Baumgart [4] more than three decades ago. Cippolla and Blake [7] showed that by analysing silhouette deformations local surface curvature can be computed along the corresponding contour generators. Forsyth [8] showed that outlines of algebraic surfaces completely determine their projective geometry from a single view. Cross et al. [9] studied the projective relationship between the coefficients of quadratic algebraic surfaces and the coefficients of the corresponding 2D algebraic silhouettes. Due to perspective projection, the relationship between algebraic surface and algebraic plane curve coefficients is very complex for higher-order surfaces. Kang et al. [18] reconstructed 3D surfaces from occluding contours of algebraic surfaces using a linear dual-surface approach that makes use of the duality between 3D points and tangent planes.

For 2D-3D pose estimation, Kriegman and Ponce [20] parameterised image silhouette equations by 3D pose parameters and minimized the distance between such equations and pixels representing image outlines to obtain the optimal pose. Rosenhahn [28] used the explicit approach instead and back-projected lines through the silhouette pixels in order to register 3D models with those lines. He extended approach to human motion tracking in [29]. Ilic et al. [16] and Knossow et al. [19] also used image silhouettes for human motion tracking using implicit equations.

There are also several variations in the methods of pose estimation. An overview of existing techniques for pose estimation is given by J.S. Goddards PhD-thesis [17].

3 POSE ESTIMATION WITH ENTITIES CORRESPONDENCE

3.1 Pose constraints in conformal geometric algebra

In this section we give a brief framework about how the interaction of entities in geometric algebras are applied on the pose problem. As mentioned earlier, the main problem in the pose estimation is determination of the 2D image features corresponding to 3D object features. The constraint equations can lead to equations of the following equation [28] (this one is just for point correspondences).

$$\lambda\left(\left(M\underline{X}\tilde{M}\right)\underline{\times}e_\infty\wedge(O\wedge x)\right)\cdot e_+ = 0 \qquad (1)$$

where λ is a scale parameter, O is the camera position, the underline characters stand for the points in conformal space, the commutator $\underline{\times}$ [26] is used to model a distance measure.

The interpretation of the equation is simple as the equation can be separated in the following manner,

$$\lambda\underbrace{\underbrace{\left(\underbrace{\left(M\underbrace{\underline{X}}_{\text{point in conformal}}\tilde{M}\right)}_{\text{rigid motion}}\underline{\times}e_\infty\wedge\underbrace{\left(\underbrace{O}_{\text{optical center}}\wedge\underbrace{x}_{\text{image point}}\right)}_{\text{projection ray in conformal space}}\right)}_{\text{collinearity of the object point with reconstructed line}}\cdot e_+}_{\text{Euclidean distance measure between line and point}} = 0 \qquad (2)$$

We see that the strategy of expressing the pose problem can directly be seen from the equation. All geometric aspects are considered and the equation is compact and easy to interpret.

The main denotes advantages of the constraint equations are:

1. The constraints are expressed in a multiplicative manner, they are concise and easy to interpret. This is the basis for further extensions, like kinematic chains and other higher order algebraic entities.

2. The whole geometry within the scenario is concerned and strictly modeled. This ensures an optimal treating of the geometry and the knowledge that no geometric aspects have been neglected or approximated which is sometimes done in the literature [5] by using orthographic camera models.

3.2 Numerical estimation of pose parameters

In the section 3.1, we give constraint equation that relate 3D object entities to 2D image information. In these equations the object, camera and image information are assumed to be known, the motor M expressing the motion is assumed to be unknown. The main question is now, how to solve a set of constraint equations for multiple features with respect to the unknown motor M. Since a motor is a polynomial of infinite degree, this is a non-trivial task, especially in the case of real-time estimation.

How to get a linear equation with respect to the generators of the motor? We try to solve this problem with exponential representation of motors and the Taylor series expansion with the first approximation order. This leads to a mapping of the above mentioned global motion transformation to a twist representation, which allows for incremental changes of pose. This results in linear equations in the generators of the unknown 3D rigid body motion. In this section the linearization of the motor is derived. For simplicity, we consider the case of point transformations.

The Euclidean transformations of a point X in conformal space caused by the motor M is approximated as:

$$\begin{aligned} M\underline{X}\tilde{M} &= \exp\left(-\frac{\theta}{2}\left(l'+e_\infty m'\right)\right)\underline{X}\exp\left(\frac{\theta}{2}\left(l'+e_\infty m'\right)\right) \\ &\approx \left(1-\frac{\theta}{2}\left(l'+e_\infty m'\right)\right)\underline{X}\left(1+\frac{\theta}{2}\left(l'+e_\infty m'\right)\right) \\ &\approx E+e_\infty\left(x-\theta(l'\cdot x)-\theta m'\right) \qquad (3)\end{aligned}$$

We assume $l := \theta l'$ and $m := \theta m'$, then:

$$M\underline{X}\tilde{M} \approx E+e_\infty(x-l\cdot x-m) \qquad (4)$$

In the next step we estimate the motion of the 3D object with the previously derived point-line constraint, it leads to

$$\begin{aligned} 0 &= M\underline{X}\tilde{M}\underline{\times}\underline{L} \\ 0 &= \exp\left(-\frac{\theta}{2}\left(l'+e_\infty m'\right)\right)\underline{X}\exp\left(\frac{\theta}{2}\left(l'+e_\infty m'\right)\right)\underline{\times}\underline{L} \\ 0 &\approx \left(E+e_\infty(x-l\cdot x-m)\right)\underline{\times}\underline{L} \\ 0 &= \lambda\left(E+e_\infty(x-l\cdot x-m)\right)\underline{\times}\underline{L} \qquad (5)\end{aligned}$$

Due to the approximation $\approx$ in equation (5), the unknown motion parameters l and m are linear. This equation contains six unknown parameters for the rigid body motion. The unknowns are the unknown twist parameters for the motion. In the last step the linearized constraints are scaled with a suitable factor λ to express an Euclidean distance measure as mentioned in section 3.1. This means, all transformations are done in the conformal space, only in the last step the constraint equations are scaled for transformations in the Euclidean space.

The linear equations are solved for a set of correspondences by applying the Householder method [27]. From the solution of the system of equations, the motion parameters R, t can easily be recovered by evaluating $\theta := \|l\|, l' := \frac{l}{\theta}, m' := \frac{m}{\theta}$. The motor M can be evaluated by applying the Rodrigues' formula.

The principle of this approximation is illustrated in figure 1. The aim is to rotate a point $\underline{X}$ by 90 degrees to a point $\underline{X}'$. The first order approximation of the rotation leads to the tangent of the circle passing through X.

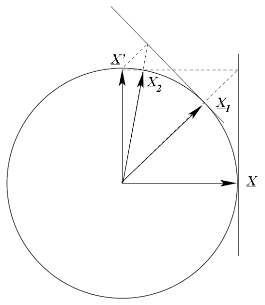


Figure 1: Principle of convergence for the iteration of a point X rotated around 90 degrees to a point X'. X_1 is the result of the first iteration and X_2 is the result of the second iteration. [28]

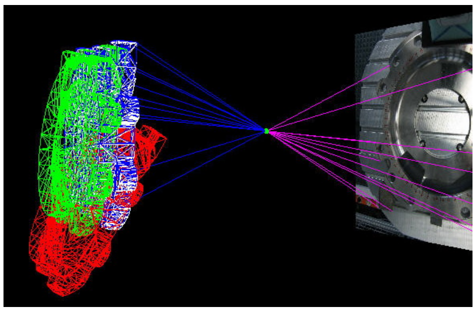


Figure 2: Iterative pose estimation process, the red object is the position after the first iteration, the green one is after the second iteration, the blue and the white objects are the positions after the third and fourth iteration.

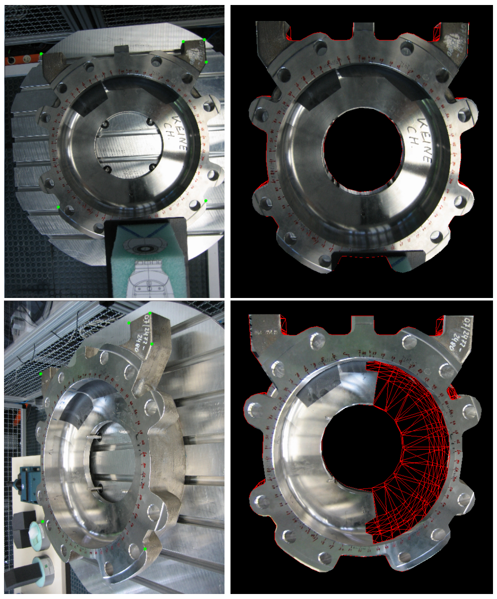


Figure 3: Left: 2D image, the green points are the corresponding points. Right: final textured 3D object.

Normalizing the tangent line to $\underline{X}'$(denoted by dashed lines) $\underline{X}_1$ is gained as the first order approximation of the required point $\underline{X}'$. By repeating this procedure the points $\underline{X}_2 \ldots \underline{X}_n$ will be estimated, approaching to the point $\underline{X}'$. It is clear from figure 1 that the convergence rate of a rotation depends on the amount of the expected rotation.

All angles converge during the iteration. For the most cases just a few iterations are sufficient to get a good approximation. In situations where only small rotations are assumed, four iterations are sufficient for all cases.

3.3 Result

We use the point-line constraint to construct the linear equation matrix and the least square method to solve it. Four iterations are needed to compute the final translation t and rotation R. The developed algorithm interactively computes the camera position (see Figure 2).

The next problem that we need to solve for texture mapping is detection of visible areas. We should detect visible triangle for the computed camera position. To solve the problem, the Ray-tracing algorithm is used: For each point, there exists a ray from this point to the camera position. If the ray hits a triangle of the object before it gets to the camera position, we consider the point is invisible, otherwise it is visible. If three points of the triangle are visible, we consider this triangle as visible. Ray-tracing algorithm gives the texture coordinates in the 2D image, which are used in texture mapping. There're some experiments results in Figure 3, on the left are the original image, the green points are the corresponding points between 2D image and 3D object detected by user, on the right are the texuted object. We see clearly that the developed algorithm successfully solve the texture mapping problem.

Now we present a practical application of the developed algorithm for non-destructive testing (NDT). Thermal inspection is one of the numerous methods in NDT. The inspection consists of two cases: 1) excitation using the flash lamps 2) observation of the cooling process using an infrared camera. The existing methods in representation of infrared data involve 1D (time profile) and 2D (x/y space) forms. They're caused by using focal plane array (FPA) detectors in infrared cameras. The technique developed in this paper significantly extends capabilities in representation of acquired data. The combination with prior known geometry of an object to be inspected makes the representation more informative and allows analyzing the physical processes inside the object taking into account its geometry. The developed algorithm was successfully tested for visualization of thermal inspection data, Figure 4 shows the infrared 3D data sequence visualization as time increase.

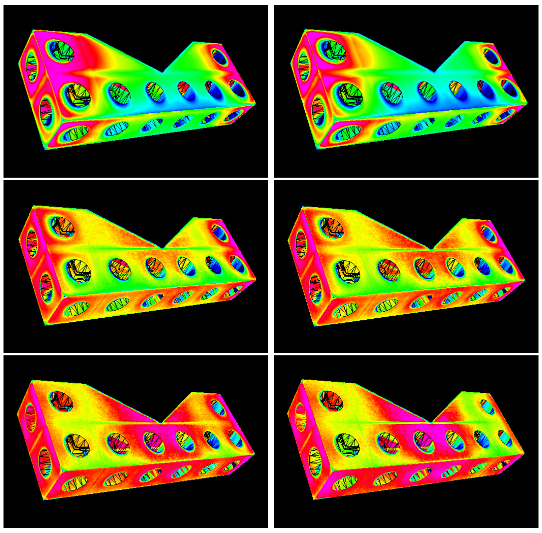


Figure 4: From Left to right, from top to bottom: 3D Visualization of the thermal image set.

4 POSE ESTIMATION WITH ACTIVE CONTOUR CORRESPONDENCE

4.1 3D object contour in Fourier domain

This section we intorduce signal theoretic foundations. The aim is to define the discrete Fourier transformation and its extension to the 3D space in classical matrix calculus. More detail information can be found in [3, 2].

For 3D contour interpolation a set $f_j^3 \in \Re^3$ of 3D values is assumed $j = 0,\dots,M-1, M \in \aleph$. These values are contour points of a closed contour. To achieve a 3D contour interpolation, the 3D signal can be interpreted as 3 separate 1D signals:

$$F_m^3 = \frac{1}{M}\sum_{u=0}^{M-1}\begin{pmatrix} f_u^3(1) \\ f_u^3(2) \\ f_u^3(3) \end{pmatrix} \exp\left(\frac{-2\pi ium}{M}\right) \quad (6)$$

And its inverse transformation can be written as

$$f_u^3 = \sum_{m=0}^{M-1}\begin{pmatrix} F_m^3(1) \\ F_m^3(2) \\ F_m^3(3) \end{pmatrix} \exp\left(\frac{2\pi imu}{M}\right) \quad (7)$$

Taking only a subset of the phase vectors leads to a low-pass approximation of the contour. This is applied to speed up the algorithm for pose estimation of free-form contours and to avoid local minimum during iterations.

The user should give initial position of the 3D object firstly, and then select the region or sub-region of the 3D object which is captured in the 2D image. Finally the selected object will be mapped to 2D image, as Figure 5 left shows. This 2D information can help us to find the discrete contour points of the 3D object. The contour points of the mapped 2D image correspond to the discrete contour points of the 3D object. As Figure 5 middle shows, the algorithm can find the discrete contour points. With Fourier transformation as talked above, we can get the continuous contour of the 3D object from the discrete contour points (as Figure 5 right shows). The user can also select the sub-region of the 3D object and with the same processing, get the continuous contour of the 3D object.

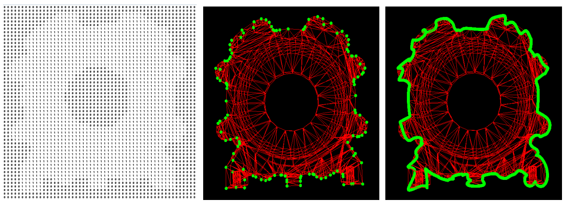


Figure 5: Left: Mapped 2D image from 3D object. '1' is the object region, '0' is the background region. Middle: Discrete contour points of the 3D object. Right: Continuous contour of the 3D object.

4.2 2D image contour

2D active image extraction algrithom [6] is proposed an active contour model based on Mumford-Shah segmentation technique and the level set method. The model is not based on an edge-function to stop the evolving curve on the desired boundary. Also, we do not need to smooth the initial image, even if it is very noisy and in this way, the locations of boundaries are very well detected and preserved. By this model, we can detect objects whose boundaries are not necessarily defined by gradient or with very smooth boundaries, for which the classical active contour models are not applicable. The position of the initial curve can be anywhere in the image, and it does not necessarily surround the objects to be detected.

Let us define the evolving curve C in Ω, as the boundary of an open subset ω (i.e. $\omega \subset \Omega$, and $C = \partial\omega$). Then, $inside(C)$ denotes the region ω, and $outside(C)$ denotes the region $\Omega\backslash\omega$. This method is the minimization of an energy based-segmentation. Let us first explain the basic idea of the model in a simple case. Assume that the image u_0 is formed by two regions of approximatively piecewise-constant intensities, of distinct values u_0^i and u_0^o. Assume further that the object to be detected is represented by the region with the value u_0^i. We denote its boundary initially by C_0. Then we have $u_0 \approx u_0^i$ inside the object (or $inside(C_0)$), and $u_0 \approx u_0^o$ outside the object (or $outside(C_0)$). Now let us consider the following 'fitting' energy function:

$$\begin{aligned} F(c_1,c_2,C) = \quad & \mu L(C) + \nu A(in(C)) \\ & +\lambda_1 \int_{in(C)} |u_0(x,y)-c_1|^2 dxdy \\ & +\lambda_2 \int_{out(C)} |u_0(x,y)-c_2|^2 dxdy \quad (8) \end{aligned}$$

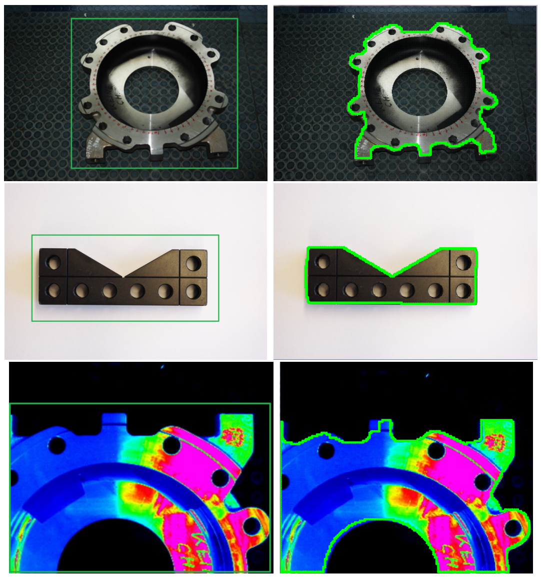


Figure 6: Left: 2D image and the initial contour given by the user (green line). Right: 2D image contour given by the active contour algorithm (green line)

Where $L(C)$ stands for the length of the contour, $A(in(C))$ stands for the area in the contour, c_1 and c_2 is are the average intensity levels inside and outside of the contour. $\mu, \nu, \lambda_1, \lambda_2$ are the weight parameters. There're some algorithms to find the minimization of the energy function, Therefore, we consider the minimization problem:

$$\inf_{c_1,c_2,C} F(c_1,c_2,C) \tag{9}$$

Simply we can consider the Euler-Lagrange equation to solve this problem. With this method the contour of the object can be extracted reiteratively. THe final results are shown in figure 6, on the left are the original images, the green lines are the initial contour defined by user, on the right are the contour results with the green line showed.

4.3 Pose estimation between 2D image and 3D object's contour

The aim is to formulate a 2D-3D pose estimation algorithm for any kind of free-form contour. The assumptions are the following:

- The object contour curve is given as a set of 3D points f_j^3, spanning the 3D contour.
- In an image of a calibrated camera, the object is observed in the image plane and a set of 2D points x_j^2 spanning the 2D contour is extracted.

Since the number of contour points in the image is often too high (e.g. 800 points in the experimental scenario), just every kth point (e.g. $k \in 5, \dots, 20$) is used to get an equal sub-sampled set of contour image points.

Note that there is no knowledge which 2D image point corresponds to the 3D point of the interpolated 3D model contour. Furthermore, a direct correspondence does not generally exist since the contours are mostly sampled from different starting points and the number of image and object points may also vary.

Using the approach for pose estimation of point-line correspondences, the Iterative Closest Point (ICP) [30] algorithm for free-form contours consists of iterating the following steps:

Algorithm 1: Normal ICP Algorithm

1. Reconstruct projection rays from the image points.
2. Estimate the nearest point of each projection ray to a point on the 3D contour.
3. Estimate the pose of the contour with the use of this correspondence set.
4. Goto 2.

The idea is that all image contour points simultaneously pull on the 3D contour. This is the normal ICP algorithm, and there exist two aspects to improve the performance.

- With Fourier transformation, increasing degree method can improve the calculation speed.
- We can improve normal ICP to avoid the local minimum problem.

We talk about the methods in detail. Increasing degree method: Using the Fourier coefficients for contour interpolation works well, but the algorithm can be made faster by using a low-pass approximation for pose estimation and by adding successively higher frequencies during the iteration. We call this technique the increasing degree method. Therefore the pose estimation procedure starts with just a few Fourier coefficients of the 3D contour and estimates the pose to a certain degree of accuracy. Then the order of used Fourier coefficients is increased and the algorithm proceeds to estimate the pose with the refined object description. Improve ICP to avoid the local minimum: We can define the error, which is sum of the distances between the posed object points and the nearest rays from the image points. The user can define a threshold, for our experiments, we define the threshold 0.005. If the error is bigger than the threshold, the ICP comes to the local minimum. Then rotate the image space, such as 10 degree around the view angle, then do the normal ICP again, do this procedure again and again, until the ICP get the error smaller than the threshold. The algorithm pipeline is as follows:

With the improved ICP algorithm, the performance of pose estimation results are presented in Figure 7.

Algorithm 2: Improved ICP Algorithm

```
1. If error > threshold (0.005), Rotation of the image.
   (10 degree around the view angle).
2. Pose estimation, do this step 4 times
   (a) Reconstruct projection rays from the image points.
   (b) Estimate the nearest point of each projection ray
       to a point on the 3D contour points, which is produced
       by the Fourier interpolation.
   (c) Estimate the pose of the contour with the use of
       this correspondence set.
   (d) Increasing the Fourier coefficients of the 3D object
       contour, goto (b).
3. Calculation of the new error. Goto 1.
End
```

5 CONCLUSION

The main focus concentrates on pose estimation based on Geometric Algebra and 3D data visulazation with texture mapping. 3D object models are treated feature based and active contour form based: The results of this paper are summarized in the following points:

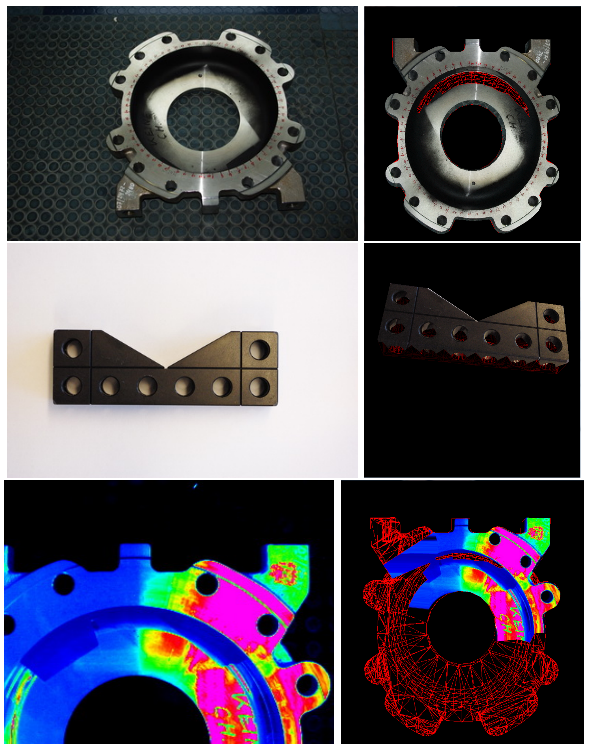


Figure 7: Left: 2D image. Right: textured 3D object after contour corresponding estimation.

- The geometry of the 2D-3D pose estimation scenario is analyzed and the interaction of entities given in conformal space. It leads to a compact and linear description of the pose problem which contains a distance measure. These equations can further be scaled by a scalar which allows for an adaptive weighting of the constraints. The constraint equations are solved by linearizing and iterating the equations. The estimation of pose parameters is high performance.

- The approach for modeling curves is related to model 3D contours by using Fourier descriptors. In this context ICP algorithms are used to estimate the correspondences and poses for image contours and object contours. The use of low-pass information enables one further to avoid local minimum and to speed up the algorithm. Furthermore, an automatic method avoid the local minimum is possible, which stabilizes the pose results.

The next extension of contour based free-form pose estimation is pose estimation of free-form surfaces. This has a much higher degree of complexity, similar to the extension of the 1D analytic signal and 1D quadrature filters to 2D in an isotropic way, as presented in [14].

Though the ICP algorithm works fine and stable in tracking situations, its computational overhead leads to hardly realizable real-time systems for complex object models. Especially for the camera calibration, it's difficult to realizable a stable and real-time performance. Here also some work is possible and promising. E.g. no fast Fourier transformation is applied so far and the minima-search in the gradient descent method is highly parallelizable. But maybe new search strategies are better suited than the used ICP algorithm.

Another extendable topic is the image processing for pose estimation. So far easy scenarios are assumed, e.g. with little background noise. The image processing is kept simple to extract the image contour, since the geometric aspects of the pose scenario are dealt with in this paper.

This leads to further extensions for computer graphics or geometric algebra and is an interesting topic for future research.

REFERENCES

[1] H. A. aposujo, R. L. Carceroni, and C. M. Brown. A fully projective formulation for lowe aposujo tracking algorithm. Technical report, 1996.

[2] J. B. Digitale bildverarbeitung. *Springer-Verlag, Berling, Heidelberg, New-York*, 1997.

[3] K. B. Fouriertransformation fuer ingenieur- und naturwissenschafte. *Springer-Verlag, Berlin, Heidelberg, New York*, 2001.

[4] B. Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, stanford University, 1974.

[5] C. Bregler and J. Malik. Tracking people with twists and exponential maps, 1998.

[6] T. F. Chan and L. A. Vese. Active contours without edges, 2001.

[7] R. Cipolla, G. Fletcher, and P. Giblin. Surface geometry from cusps of apparent contours. In *In: ICCV*, pages 858–863, 1995.

[8] F. Computer and D. A. Forsyth. Recognizing algebraic surfaces from their outlines. In *In International Conference on Computer Vision*, pages 476–480, 1992.

[9] G. Cross and A. Zisserman. Quadric surface reconstruction from dual-space geometry. *In Proceedings of the 6th International Conference on Computer Vision,Bombay, India*, pages 25–31, January 1998.

[10] K. Daniilidis. Hand-eye calibration using dual quaternions. *International Journal of Robotics Research*, 18:286–298, 1998.

[11] L. D.G. Solving for the parameters of object models from image descriptions. *in Proc. ARPA Image Understanding Workshop*, pages 121–127, 1980.

[12] L. D.G. Three-dimensional object recognition from single twodimensional images. *Artificial Intelligence*, Vol. 31, No. 3:355–395, 1987.

[13] L. Dorst. *Geometric Algebra for Computer Science An Object -Oriented Approach to Geometry, Chapter 23.* 2008.

[14] FelsbergM. Low-level image processing with the structure multivector. Technical report, Technical Report 0203,Christian-Albrechts-Universitaet zu Kiel, Institut fuer Informatik und Praktische Mathematik, 2002.

[15] D. Huber and M. Hebert. Fully automatic registration of multiple 3d data sets, 2001.

[16] S. Ilic, M. Salzmann, and P. Fua. Implicit surfaces make for better silhouettes,cvpr 05, 1135.

[17] G. J.S. *Pose and Motion Estimation From Vision Using Dual Quaternion-Based Extended Kalman Filtering.* PhD thesis, Springer-Verlag, New York Inc, 2001.

[18] K. Kang, J.-P. Tarel, R. Fishman, and D. Coope. A linear dual-space approach to 3d surface reconstruction from occluding contours using algebraic surfaces. In *In International Conference on Computer Vision*, pages 198–204, 2001.

[19] D. Knossow, R. Ronfard, R. Horaud, and F. Devernay. F.: Tracking with the kinematics of extremal contours. In *Computer Vision ¨C ACCV 2006. LNCS*, pages 664–673. Springer, 2006.

[20] D. J. Kriegman and J. Ponce. On recognizing and positioning curved 3-d objects from image contours. *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 12:1127–1137, Dec 1990.

[21] N. Krueger, M. Ackermann, G. Sommer, and L. F. K. Systeme. Accumulation of object representations utilizing interaction of robot action and perception. *Knowledge Based Systems*, 13:200–2, 2002.

[22] G. W. E. L. *Object Recognition by Computer*. The MIT Press,Cambridge, MA, 1990.

[23] H. D. Li H. and R. A. A. generalized homogeneous coordinates for computational geometry. pages 27–52, 2001.

[24] W. M.W. and S. L. Estimating 3-d location parameters using dual number quaternions. *Computer Vision, Graphics, and Image Processing (CVGIP): Image Understanding*, Vol. 54, No. 3:358¨C367, 1991.

[25] F. O. *Three-Dimensional Computer Vision, A Geometric Viewpoint.* MIT Press, Cambridge, 1993.

[26] C. Perwass. *Applications of Geometric Algebra in Computer Vision.* PhD thesis, Cambridge University, 2000.

[27] V. W. Press W.H., Teukolsky S.A. and F. B.P. *Numerical Recipes*. Cambridge University Press, 1994.

[28] B. Rosenhahn. *Pose Estimation Revisited.* PhD thesis, Inst. f. Informatik u. Prakt. Math. der Christian-Albrechts-Universit"at zu Kiel, 2003.

[29] B. Rosenhahn, U. G. Kersting, A. W. Smith, J. K. Gurney, T. Brox, and R. Klette. A system for marker-less human motion estimation. In *Page 6 [22] Rosenhahn B., Klette*, pages 45–51. Springer, 2005.

[30] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *INTERNATIONAL CONFERENCE ON 3-D DIGITAL IMAGING AND MODELING*, 2001.

[31] B. W. *Kinematik und Quaternionen, Mathematische Monographien 4.* Deutscher Verlag der Wissenschaften, 1960.

[32] M. Zerroug and R. Nevatia. Pose estimation of multipart curved objects. *Image Understanding Workshop (IUW)*, pp:831–835, 1996.

A GPU-Supported Approach to Registration of 3D Scan Data

Adrie Kooijman
Delft University of Technology
Design Engineering dept.
Landbergstraat 15
2628CE, Delft, the Netherlands
a.kooijman@tudelft.nl

Joris Vergeest
Delft University of Technology
Design Engineering dept.
Landbergstraat 15
2628CE, Delft, the Netherlands
j.s.m.vergeest@tudelft.nl

ABSTRACT

We present a method of partial matching two shapes aimed to support 3D scanning applications. To digitize the surface of a 3D object, a number of scan views are taken from different angles, collectively representing the entire surface of the object. The registration process assigns a transformation to each of the scan views such that the surface of the object can be reconstructed. The motivation of our work is to provide an initial geometric model to a designer who develops the shape further using Computer-Aided Design. We propose a method of pairwise shape matching of partial scans, without depending on any distinct geometric feature type. The degree of matching of two shapes, that is the congruence of their overlapping portions, has been defined by a matching index *I*, which should be maximized to attain the right relative placement of the two shapes. By using a simplification of one of the shapes and a sampling strategy in 6D configuration space we achieve matching in an amount of time that would be too high for practical interactive use. However, we offload the most computation intensive part of the matching process to the graphical processing unit (GPU) to achieve 50-80 times shorter times. The method does not require the detection of intrinsic or artificial features (or shape descriptors), nor optical markers nor position or orientation tracking information. To our best knowledge there are no previous reports about practically usable results of similar methods. We present the outcomes of tests with scanned point clouds and initial conclusions about the robustness and costs of the method. The limitations of the method are discussed and improvements of the search strategy and the GPU-based computation of the matching index are proposed.

Keywords: Partial shape matching, 3D scanning, automatic surface registration, brute force, configuration space sampling, GPU

1. INTRODUCTION

Surface registration is an essential part of 3D scan data processing. To digitize the surface of a 3D object, a number of scan views are taken from different angles. The scan views should partially overlap and collectively represent the entire surface of the object. The registration process assigns a transformation to each of the scan views such that they all get aligned, and the surface of the object can be reconstructed. Typically, the registration process is supported by a software package that is provided by the scanner manufacturer, and which should be operated by the user. It starts with pairwise registration of the surfaces, where the user should designate several pairs of corresponding points in the two surfaces. This information is sufficient for the software to estimate the global alignment of the two surface (initial pose), and the registration is then accomplished by method based on the Iterative Closest Point (ICP) algorithm [Besl and McKay 1992]. Finally a global registration (sometimes called fine registration) slightly adjusts all surfaces such that they optimally represent the object's surface. Often the users of the scanning software are incidental users: a scan is made, and the result is exported to a CAD software package for further processing. Since the user-assisted registration process is often perceived as tedious and sometimes even error-prone for incidental users, much research is, until today, devoted to the development of methods to achieve automatic registration that require no or minimal user intervention. One approach is to find the initial transformation by supplying additional data about the scan view orientations and/or positions from a (possibly wireless) tracking device or from mechanical manipulators attached to the object or scanning device. Another way to obtain this initial pose estimation is by physically placing reflective markers on the object surface and to use these markers to track the position of the object. The correct working of the hand-held scanning device named Handyscan 3D [Creaform 2009] depends on the occurrence of such markers. Most methods of initial pose estimation are based on the detection of local shape descriptors in two surfaces. From the placements of these shape descriptors, an approximate transformation *M* can be calculated to achieve an initial alignment of the two surfaces, see for example [Li and Guskov 2005, Pottman *et al* 2007]. In case the local descriptors are relatively small, they provide only a limited robustness of the initial pose. The method proposed by [Aiger *et al* 2008] is based on wide structures, and therefore more accurate. A general draw-back of shape features is their possible non-uniqueness and their possible occurrence outside of the overlap region, such that the number of features inside the overlap gets too small to derive the initial pose transformation. Finally we mention approaches based on restrictions imposed on the scan views taken. Here the user (the person who takes the 3D scan views) is instructed to take the scan views in a particular manner. For example [Wang *et al* 2007] could achieve automatic surface registration of scans of a hand-held object, where the object was manually rotated about a fixed axis. In fact the axis was slightly different for each following pose, since it was very difficult for a human to bring the object in subsequent orientations about a common axis. However, in practice it turned out possible to predict the pose of a scan view relative to the previous one accurately enough to apply ICP successfully.

The motivation of our work is to support 3D scanning of object surfaces, where the objects are typically of size between 100 and 500mm. The target application is providing an initial geometric model to a designer who develops the shape further using Computer-Aided Design (CAD). In industrial design engineering, product design can commence from a hand-made clay model, which is scanned at a particular stage of shape development after which the scanned model is further processed in the computer. To make 3D scanning acceptable by industrial designers and artists, the problem of automatic scan registration should be solved. However, it has turned out acceptable for the users to put some simple restrictions to the way of scanning e.g by limiting the rotation angle and direction between scans to achieve semi-automatic scan registration.

In this paper we propose a method of pairwise shape matching based on direct comparison of the two shapes, without depending on the occurrence of any geometric feature type or other references. The method is described in Section 2, the implementation in Section 3 and results of initial tests are presented in Section 4. In Section 5 we evaluate the method and discuss its limitations and opportunities for improvement. Because the method is very computational intensive a large part of the computations are performed parallel on a graphics card. By executing several hundreds of calculations in parallel the total calculation times remain acceptable for interactive use of the method.

The main contribution of this work is the improved understanding of the role the GPU can play in 3D shape alignment processes. The acceleration is explored for the basic method described in Section 2, and it is expected that the increase in performance can also be obtained for dedicated methods, as will be discussed in Section 5.

2. DESCRIPTION OF THE METHOD

The problem of matching two shape representations *A* and *B* is to find a rigid transformation *M* such that *MB* and *A* are have the correct relative placement. We assume that *A* and *B* each represent a portion of a surface of a 3D object, and that there may exist subsets of *A* and of *B* representing the same portion of that object, called the overlap region. If we can detect portions of *A* and of *B* which are congruent (or approximately congruent in case the representations are not exact), then we have candidate overlaps and we can compute the transformation *M*. We propose a method of pre-alignment based on explicit sampling of the 6D configuration space of *M* and a shape matching index *I* (*A*, *MB*) representing the degree of consistency of shapes *A* and *MB*. Our approach is based on the raw geometric data *A* and *B* only, that is on the point sets representing the shapes; no surface generation or triangulation is required. The only preprocessing we do is down-sampling of point cloud *B* to reduce the amount of calculation without hampering the effectiveness of the method. Then we compute the distance of that reduced point set to point cloud *A* as:

$$I(A, MB) = \frac{1}{n_B} \sum_{i=1}^{n_B} (c + d_i^{\,2})^{-1}, \text{ whith}$$

$$d_i = \min_{j=1,\,n_A} |b_i - a_j| \ , \qquad (1)$$

where a_j and b_j are points in point cloud *A* and (down sampled) point cloud *MB*, respectively. The damping term c can be set to any positive value; it will influence the shape of the distribution of *I* but not the location of its maximum. If point sets *A* and *B* are identical, then *I* can reach 1/*c*. If *A* and *B* are not identical but each contain points representing the same surface then *I* will typically reach 1/(*c* + *s*), where *s* the scan resolution of *A*. The basic assumption of our method is that a substantial overlap of *A* and *MB* provides a big contribution to *I*, whereas points outside the overlap region contribute only little. At the end of this paper we will discuss possible alternative measures. The basic assumption of our method is that a substantial overlap of *A* and *MB* provides a big contribution to *I*, whereas points outside the overlap region contribute only little. The index is relatively large if many points in *B* are close to any point in *A*. The index indicates, therefore the degree of overlap of *MB* and *A*. If fraction *p* of the extent of *MB* is overlapping with *A* then *I* can reach *p*/(*c* + *s*). Down sampling of point cloud *B*, in our experiments, is typically from about 10,000 to 250, where we applied a simple $G{\times}G{\times}G$ grid to divide the bounding box of *B*, typically with *G*=10. We selected one point from *B* for each grid element, rather than the average of points inside a grid element, since the latter position could be significantly off *A* even at perfect alignment. Point cloud *A* was always left intact; no simplification or reduction was applied as to achieve a good estimate of distance d_i.

The numerical approach to sample in configuration space of *M* is as follows. $M = M(\alpha, \beta, \gamma, x, y, z)$ transforms point cloud *B* into *MB*, which is rotated by the three Euler angles [Craig, 1989], followed by a shift amounting to vector $(x, y, z)^T$. The amounts of shift and of orientation are chosen, and typically around 10mm and 5 degrees, or less, depending on the dimensions of the object and the refinement stage of the searching algorithm. The shift range in a particular coordinate direction is determined by the lengths of the bounding boxes of *A* and the (rotated) *B*; the range of orientation can be limited to a practical value of for example +/–30 degrees. These values depend on the assumptions reasonable for the scanning application, discussed later. Even with these relatively large step sizes, the number of matching index computations reaches a million or more.

We have investigated the usability of *I* from two respects: 1) the goodness for detecting overlap regions and 2) efficiency with regard to interactive application. We present preliminary results here, based on a small number of scanned objects.

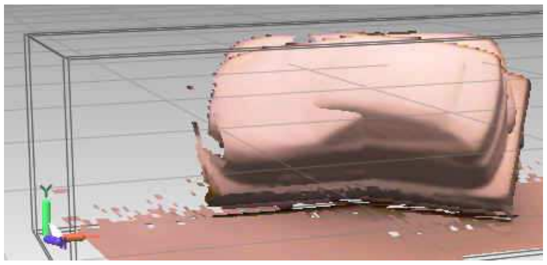


Figure 1: Two scan views of a car model

In Figure 1 two overlapping scan views of a simple foam car model are shown. The number of points in *A* and *B* are 6107 and 6518, respectively. Using 16 steps over 40 degrees for the three rotation axes and 16 steps for the three translation directions over the bounding boxes of *A* and *B*, *I* was computed 16^6 = 16.8 million times. The highest *I* produced the correct *M* and hence registration of the two scan views, found in a few minutes of computation time.

3. IMPLEMENTATION

The calculation of *I*(*MB*) is a very computation intensive job. The contribution of each point of the down-sampled point cloud *MB* is independent from the contribution by the other points. This allows parallelizing the process. Therefore we have implemented parts of the computation parallel on a Graphical Processing Unit (GPU). We used the Nvidia Geforce GTX 280 graphics card. This card contains 30 multiprocessors, each of which consists in its turn eight scalar processors. In total 240 processor cores, running at 1296MHz, are available. Each multiprocessor can handle a maximum of 512 threads and has 16kbyte shared memory. The total amount of memory available in the graphics card is 1Gigabyte. The programming environment is Microsoft Visual C++ for the CPU part of the software and Nvidia Compute Unified Device Architecture (CUDA) for the Graphical Processing unit (GPU). CUDA is integrated in Microsoft Visual Studio. This combination provides an integrated development environment including emulation and debugging of the CUDA parallel kernel program.

The down sampling of *MB* to a maximum number of 512 points reduces the amount of computations without decreasing the accuracy of the method and makes it possible to calculate the contribution of all the points of *MB'* to *I* in a single thread block on the

GPU.

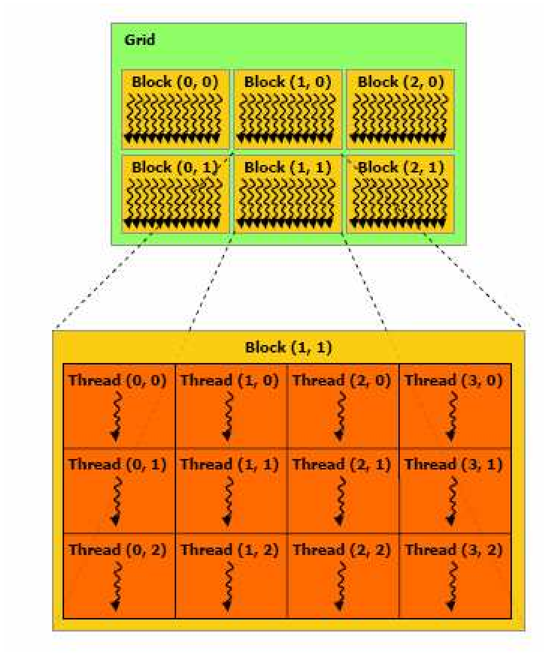


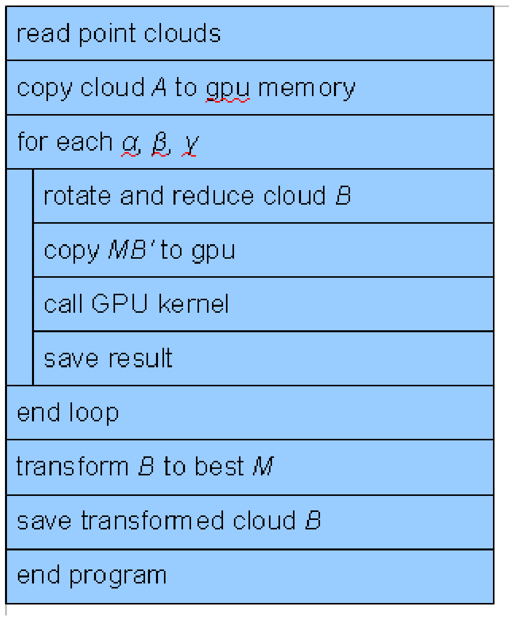
Figure 2: grid of thread blocks

The thread scheduler of the graphics card allows a 2D grid of thread blocks to be executed, where each thread block can handle a maximum of 512 threads. Figure 2, copied from the CUDA 2.0 programming manual, shows the structure of this grid of thread blocks and the threads inside such a block. In figure 3 the global flow of the CPU part of the program is shown, figure 4 shows the flow of the kernel program, which is the part of the program that is executed by the GPU. The CPU program is responsible for copying the point set data to the GPU memory and back from the GPU memory. The GPU program can not access the PC's main memory. The PC main memory is referred to as the host memory in CUDA. The GPU internal memory is known as device memory. The kernel program is set up in *X*steps * *Y*steps thread blocks, where *X*steps and *Y*steps are the number of translation steps in *X* and *Y* directions, respectively. Each thread in the kernel program has predefined constants that tell in which thread number of which thread block it is executed. This information is used to calculate the local *X*, *Y* and *Z* shift for each individual thread and to select the point from *MB'* to use. The internal thread scheduler of the GPU does the housekeeping and determines which thread block is executed at which time making the execution as efficient as possible.

From the CUDA v2.0 programming manual:

> "Given a total number of threads per grid, the number of threads per block, or equivalently the number of blocks, should be chosen to maximize the utilization of the available computing resources. This means that there should be at least as many blocks as there are multiprocessors in the device. Furthermore, running only one block per multiprocessor will force the multiprocessor to idle during thread synchronization and also during device memory reads if there are not enough threads per block to cover the load latency. It is therefore usually better to allow for two or more blocks to be active on each multiprocessor to allow overlap between blocks that wait and blocks that can run. For this to happen, not only should there be at least twice as many blocks as there are multiprocessors in the device, but also the amount of allocated shared memory per block should be at most half the total amount of shared memory available per multiprocessor. More thread blocks stream in pipeline fashion through the device and amortize overhead even more."

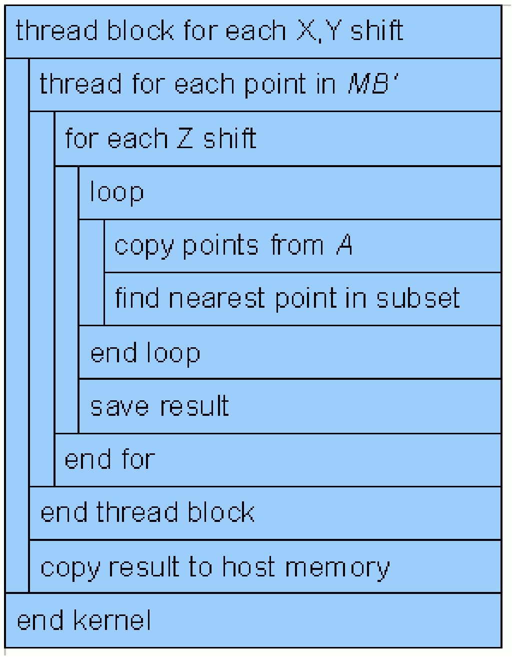
For maximum performance of the GPU kernel program it is important to submit many thread blocks. We implemented the method in such a way that for each combination of *X*- and *Y* shift a separate thread block is launched. The shift in *Z* is handled within the thread block. Each thread calculates the contribution of a single point of *MB* to *I* for all steps in Z direction. In one of our



```
read point clouds
copy cloud A to gpu memory
for each α, β, γ
    rotate and reduce cloud B
    copy MB' to gpu
    call GPU kernel
    save result
end loop
transform B to best M
save transformed cloud B
end program
```

Figure 3: CPU part of the program

experiments 16 * 16 * 16 shift steps are used, so in that case we launch 256 thread blocks which is sufficient to minimize idle times of the GPU. For each combination of *x*- and *y* shift in *M* a thread block is launched on the GPU. The *z*-shifts are handled within the individual threads. Each thread in a thread block calculates the contribution of a single point of point cloud *B* to matching index *I* for each defined *z* shift. Rotating and down sampling of point set *B* is performed on the CPU before kernel execution.



```
thread block for each X,Y shift
    thread for each point in MB'
        for each Z shift
            loop
                copy points from A
                find nearest point in subset
            end loop
            save result
        end for
    end thread block
    copy result to host memory
end kernel
```

Figure 4: GPU part of the program

The GPU card has a total of 1Gigabyte of global memory. Each multiprocessor in the GPU has 16 kilobytes of shared memory. Global memory requires up to 600 clock cycles to access while shared memory is accessible from within a thread block in only a few clock cycles. The amount of shared memory is not enough to load point set *A* completely. We copy subsets of n_B points of point set *A* from global memory to shared memory; in parallel. This means that each thread in the thread block copies one point per subset so n_B points are copied in parallel. After copying a sub set each thread calculates the distance from its designated point to all points in the sub set and keeps track of the nearest. After finding the nearest point in a sub set, the next subset is copied. This procedure is repeated until all points of *A* are tested. This way of accessing point set *A* minimizes the overhead caused by memory latency.

4. RESULTS

We have investigated the usability of *I* if equation (1) from two respects: 1) the goodness for detecting overlap regions and 2) efficiency with regard to the interactive application described above. We present preliminary results here, based on a limited number of scanned objects. The scanner applied was a Minolta Vivid700 3D digitizer, producing maximally 40,000 point measurements in one view.

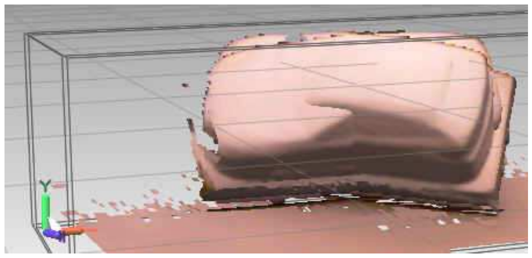
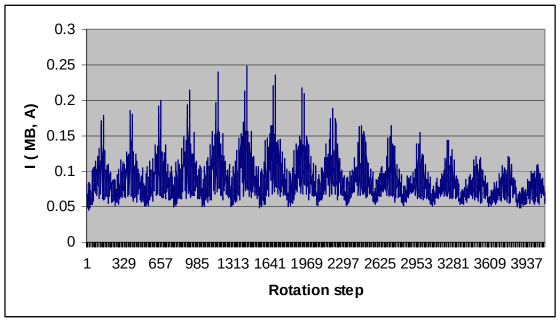


Figure 5. Two scan views with large

In figure 5, two scan views of a simple foam car model, having a maximum length of about 200mm, and their overlap region are shown. Originally the scan views are point clouds but are meshed for visibility The number of points in *A* and *B* are 6107 and 6518, respectively. After down sampling the number of points in point cloud *MB* is reduced to approximately 256, the exact number depends on *M*. Using 16 steps over 40 degrees for each of the three rotation axes and 16 steps for each of the three translation directions over the sum of the lengths of the bounding boxes of *A* and *B*, 16^6 = 16.8 million matching indices were computed. The computation time was 24 minutes. The highest matching index found was *I* = 0.249, producing the correct transformation *M* and hence registration of the two scan views (not shown here). The orientations were stepped in three nested for loops, and for each of the 4096 orientations, 4096 shifts of *B* were involved. The maximum matching index found for each of these orientations is shown in Figure 6. The 16 peaks visible in the top part of the graph represent the local maxima as a function of Euler angle β, for 16 values of angle α. The bottom of Figure 6 is a more detailed view near the maximum of the graph and shows that each of the 16 peaks contains the variation of the index as a function of angle γ.



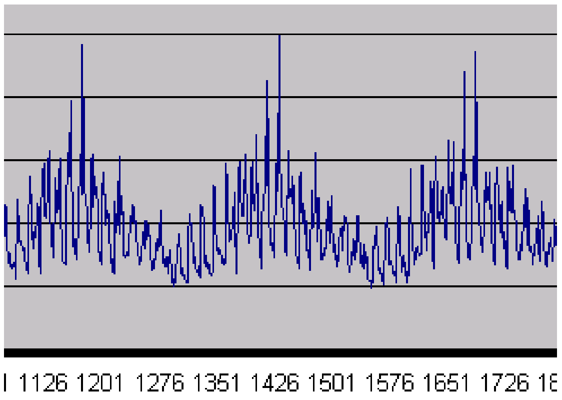


Figure 6. Top: Matching indices I(A, MB) obtained by sampling configuration space of M, plotted in order of orientations occurring in the procedure. Bottom: Detail of the graph near its maximum.

We have scanned a model puppet of Snoopy, about 200mm tall. Contrary to the car model the Snoopy model represents a much more irregular shape. Two scan views of the Snoopy model are shown in Figure 7, in arbitrary orientation, each containing about 2500 points. To test the behavior of the matching index near its maximum, we plotted the *I* (*MB*, *A*) for variations in $(x, y, z)^T$ (Figure 8). Since the overlap region is smaller, the location of the maximum matching index is less clear, compared to the case of the car model.

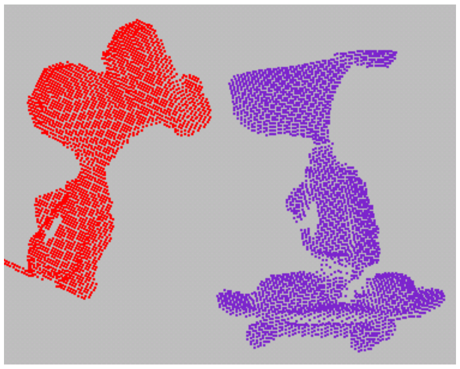


Figure 7. Two scan views of Snoopy.

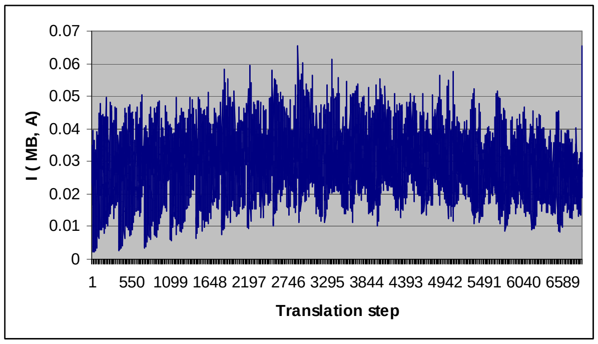


Figure 8. Distribution of the matching index near the correct matching M for deviations of the translations. The two scan views are shown in Figure 7.

We have refined the sampling procedure by repeating the sampling process, starting from the location in 6D where a maximum was found, but using smaller step sizes and ranges for the rotations and translations. In this way we iteratively approach the maximum in increasingly small steps. The scan views of the car model were used for an initial test, and in Figure 9 it can be seen that in the second stage no significant increase of *I* (*MB*, *A*) is attained, whereas in stage 3 a significant improvement is obtained the fourth stage again brings little improvement and the iteration process is aborted.

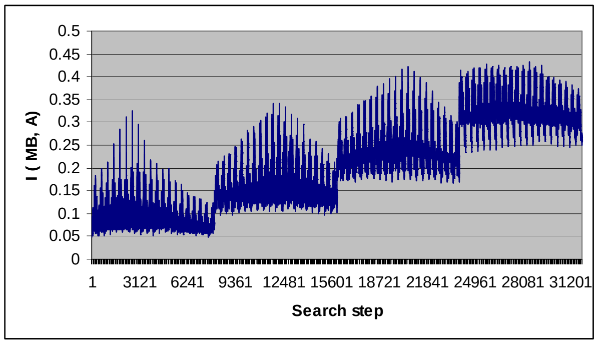


Figure 9. Matching index obtained in four refinement stages of the sampling process.

Although the number of experiments with this iterative approach was until now limited the results look very promising. If the parameters are carefully chosen the total time for the process is about the same of the first approach, but results in a better match.The current approach results in a transformation *M* that still requires fine tuning by ICP.

5. CONCLUSIONS AND IMPROVEMENTS

We have described initial experiments of pairwise surface registration based on a simple sampling strategy in configuration space. If the amount of overlap of the two surfaces is relatively large, the proposed matching index exhibits a global maximum, which can be found using the strategy in approximately 1 to 10 minutes of computation time. We have not yet achieved sampling of the full 6D configuration space with sufficiently small step size in acceptable times for interactive application. Since in that application, the user should get feedback about success or failure of the matching of the scan view just taken, the procedure should not last longer than about 10s. Using the current algorithm, the method is usable if the user would follow operation instructions (typically to exert limited rotation) to ensure that sufficient overlap remains between the last two scans. In order to release these restrictions and to speed up the calculation, several improvements are necessary.

The definition of *I* could be changed to include contributions from points in *MB* which are near *A* only. Especially when the overlap region is small this might result in a better measure, and is indeed more frequently occurring in the literature. To decrease the computation time, one could experiment with reduced *A* sets, *e.g.* and pre-sampling of configuration space. The computation of distance could be modified to let it break off when one of the *x*-, *y*- or *z*-components exceeds a threshold.

If no assumptions can be made about how the user operates the scanning system, then an fully exhaustive transformation space sampling would be required, which is infeasible, even using CUDA technology. However, near-brute force strategies can be explored relatively conveniently. An example includes the exhaustive testing of congruence of fat tetrahedrons observed in *B* for their congruence with any 4-point set in *A* [Vergeest 2009]. Without CUDA technology, such approaches would deem impractical.

REFERENCES

Aiger, D., Niloy, M., Cohen–Or, D. 2008. 4–Points Congruent Sets for Robust Pairwise Surface Registration. *ACM Trans. Graph. 27*, 3, Article 85 (August 2008).

BESL, P. J., AND MCKAY, N. D. 1992. A method for registration of 3-d shapes. IEEE Trans. on Pattern Analysis and Machine Intelligence 14, 2, 239–256.

CRAIG, J.J., 1989. *Introduction to robotics, mechanics and control.* Addison-Wesley.
CREAFORM, www.creaform.com.

LI, X., AND GUSKOV, I. 2005. Multi-scale features for approximate alignment of point-based surfaces. In Proc. Symp. Geometry Processing, 217–226.

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.

Vergeest, J.S.M., Kooijman, A., Song, Y. Partial 3D shape matching using fat tetrahedrons. Technical Report, 1 September 2009, Delft University of Technology.

Wang, H, Vergeest, JSM, Song, Y, Wiegers, T 2007. Automated 3D scan multi-view registration based on rotation estimation. In: *Proceedings of the WSCG'2007*,, University of West Bohemia, Plzen, J Rossignac and V Skala, Eds., 137-144

Raytracing Point Clouds using Geometric Algebra

Crispin Deul[1] Michael Burger[1] Dietmar Hildenbrand[1] Andreas Koch[2]

[1]Interactive Graphics Systems Group
Computer Science Department
TU Darmstadt, Germany
dietmar.hildenbrand@gris.informatik.tu-darmstadt.de

[2]Embedded Systems and Applications
Computer Science Department
TU Darmstadt, Germany
koch@esa.informatik.tu-darmstadt.de

ABSTRACT

Geometric Algebra (GA) supports the geometrically intuitive development of an algorithm with its build-in geometric primitives such as points, lines, spheres or planes. But on the negative side GA has a huge computational footprint. In this paper we study how GA can compete with traditional methods from Linear Algebra (LA) in the field of raytracing. We examine the raytracing algorithm for both GA and LA on the basis of primitive operations. Furthermore we introduce a novel framework for rendering point clouds based on spheres and planes as surface elements. We use this model to benchmark implementations of both algebras. Our results show that depending on the microprocessor architecture like CPUs, FPGAs or GPUs Geometric Algebra and Linear Algebra can raytrace with comparable speed.

Keywords: FPGA, Geometric Algebra, GPGPU, Point Cloud, Raytracing.

1 INTRODUCTION

this paper we investigate how to speed up calculations in Conformal Geometric Algebra to get comparable speed to linear algebra in the field of computer graphics. In the last decades Geometric Algebra (GA) has become a reasonable alternative in describing geometric algorithms compared to other systems like vector algebra.

One can work with Geometric Algebra in a very intuitive way since all objects of the algebra have a geometric meaning. In the conformal model, which we use throughout this paper, one can describe lines, planes and spheres directly as objects of the algebra. Furthermore operations like reflections and rotations can be applied uniformly to all geometric objects. As a result algorithms formulated with GA are very compact compared to systems describing geometry that are usually used in computer graphics.

Besides the new objects like spheres or planes and the uniformly applicable operations GA also includes many other mathematical systems like vector algebra, projective geometry or quaternions that enjoy a widespread use in computer graphics today. GA developers can slightly shift from their knowledge in these systems to the new opportunities introduced by GA while still being able to use their elaborate algorithms.

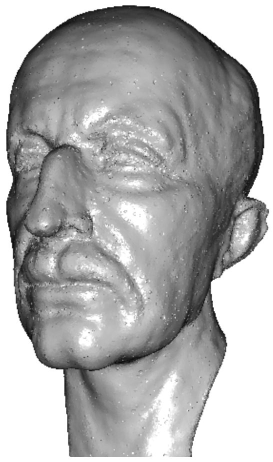


Figure 1: Max Planck point cloud rendered at 4.5 fps on an AMD HD4850 GPU. The viewport size is 640 by 480. The model consists of 96208 surfels.

While these properties of Geometric Algebra are very exciting especially for people working in graphics, computer vision or animation, there seems to be one major drawback that causes a niche existence of GA in todays applications. The mathematics in the conformal model of GA are based on the calculation of 32 dimensional so called multivectors. These multivectors represent the geometric objects of GA like spheres or planes. Different products between multivectors lead to operations like intersections or reflections. To take the scare here we have to admit that for geometric meaningful objects one often does not need more than ten non-zero entries of these multivectors. As a result mathematical effort reduces a lot in a non-naive implementation of GA [5]. While this still seems to be a lot of mathematical effort there are algorithms that work faster using GA instead of using LA [10].

There has been a second development in the last years that leads to the results of our paper. After hitting the power wall with their monolithic cores and increasing clocks CPU developers began to put more and more cores onto their chips to increase computational power. Secondly around the same time GPUs have been opened to general purpose computations by the introduction of specialized computing platforms. Today there are lots of parallel computational resources available in commodity hardware [1] [13] [16]. Geometric Algebra benefits from these architectures since multivector entries can be computed independently of each other.

In this paper we chose a field of computer graphics, namely raytracing, to investigate how much overhead there really is in choosing GA in favor of LA by simply counting the needed mathematical operations for the different raytracing primitives. We introduce a novel surfel (surface element) model based on GA spheres and planes to represent the local surface of a point cloud. With the help of this model we fortify our theoretical observations by raytracing point clouds on different microprocessor architectures. Furthermore we investigate the impact of parallelism on both Linear Algebra and Geometric Algebra versions of our algorithms.

2 RELATED WORK

The performance of raytracing with conformal GA on a general purpose CPU has already been examined in [3] and [6] but without considering the question of parallelization. Parallel FPGA implementations of raytracers were presented in [21] or [4] but only on the basis of LA. The topic of implementing a general GA processor on special hardware architectures like FPGAs was discussed in [19] and [7] with the first one only implementing one of the products of conformal GA and the second one concentrating on the 4D homogeneous space. Both also without the discussion of optimization techniques and no relation to raytracing. In this paper we try to combine all these topics to optimized GA raytracing on specialized hardware respectivly GPUs.

A number of rendering approaches and surface definitions of point clouds have been proposed in the past. Rusinkiewicz et al. [20] propose a method based on splatting small quadrats or ellipsoids onto the screen for a subset of the point cloud. Ohtake et al. [17] use local low degree implicit functions based surfels to approximate the point cloud. Their approach is close to ours since they use a surfel as a local approaximation and define the region of influence of their surfel by using a bounding sphere. Though their approach is still CPU based. A GPU based extension of the SLIM rendering is presented by Kanai et al. [14]. In contrast to our approach Kanai et al. create their primary rays by rasterizing the bounding box of the surfels. Guennebaud et al. [8] fit spheres into the point cloud similar to our approach. While we use the spheres as a direct representation of the surface Guennebauds spheres are only an intermediate step in finding an algebraic point set surface.

3 SURFEL MODEL

For our surfel model we chose a representation that directly fits to Geometric Algebra. As a result we can take advantage of the primitives and operations that are directly included in the algebra. We chose the two geometric objects plane and sphere as a basis of the surfels. These two objects are the only objects that on their own represent a surface in GA. With planes and spheres we can directly approximate local details of a real-world model. To get a representation of a whole model we simply use several of the surfels that each represent different local features of the model on their own. Since there are no data file sources for our new model representation we have to acquire the data from other representations. One way is to use point clouds as a data basis and to fit the surfel locally into a neighborhood of points. Another way would be to use triangle meshes. One could create the planes by using the plane of a triangle. Spheres could be created by using one vertex and three of its neighbors to describe a sphere of GA with the help of the outer product.

3.1 Building the Model

We build our surface by using point clouds as a data basis. An Algorithm to fit planes and spheres into point clouds has been published in [9]. The algorithm is based on the distance measure of GA between points and spheres. With the distance measure we can define a least squares approach for the point neighbourhood. Based on the least squares approach the eigenvalues of a 5x5 matrix have to be solved where the smallest positive eigenvalue directly includes coefficients of a sphere. A nice property of the algorithm is that we do not have to care whether the point neighborhood is planar since then the result of the algorithm are the coefficients of a plane. In fact in geometric algebra one can think of a plane as a sphere with infinite radius [12].

We use an iterative algorithm to get our final model. We fit surfels into the point cloud as long as there are points that are not represented by one of the already fitted surfels. To get the local neighborhood we randomly chose a non-fitted point which we call the fitting point. With the fitting point we query a kd-tree including the whole point cloud for the k nearest neighbors. The fitting point is always assumed to be fitted by the calculated surfel from the fitting algorithm. For the k neighbors we calculate the distance to the surfel. We define each neighbor to be fitted if its distance to the surfel is below a previously defined bound ε. Using this approach we can significantly reduce the data amount compared to a naive algorithm where one would fit a surfel for every point of the point cloud.

Furthermore we introduce a bounding mechanism into our surfel model. If you consider spheres with low curvature or planes these objects will usually cover the whole image while often approximating a part of the model data that is relatively small in the image space. To counter this behaviour we introduce the bounding mechanism. A natural choice for the bound is a region with some radius around the fitting point that includes most of the points in the neighborhood that were involved in the fitting process. The translation of this requirement into GA is a sphere centered at the fitting point. We calculate a first radius candidate of the bounding sphere by taking the distance between the fitting point and the farthest point of the neighborhood that is fitted by the surfel. In most cases we can take this candidate as a feasible radius for the bounding sphere but there are cases where the fitted sphere will be similar in size to the bounding sphere if taking this candidate for the radius. A result of the similar size are visual artifacts in the final rendered image. For many models it is sufficient to take a value between the radius down to a quarter of the radius of the fitting sphere as an upper bound for the radius of the bounding sphere to avoid most artifacts.

3.2 Raytracing the Surfel Model

In Raytracing we want to know where the nearest intersection between a ray shot from the camera through a pixel and the surface is located in space. We represent our rays by lines of Geometric Algebra. We create our rays by taking the camera origin and the 3D position of the pixel on the image plane. With these two points we build the outer product with the point at infinity to get a line. We can intersect the lines with both spheres and planes. Since planes in GA are spheres with infinite radius we can use our intersection indicator and intersection point algorithms for both of these objects. The algorithm to find the nearest intersection point for one pixel looks as follows:

1. **Find a candidate fit** We use both brute force and spacial data structure based approches depending on the use of our results.

2. **Intersect the bounding sphere** The bounding sphere indicates the region in space where the fitted surfel is feasible regarding the model data. Additionally in most cases the bounding sphere is much smaller in screen space than the fitted surfel. To reduce the intersection operations with fitted surfels we first calculate the intersection indicator of the bounding sphere.

3. **Test for intersection** The effort of calculating the intersection points can be saved in some cases when the ray intersects the bounding sphere but is tangential to the surfel. Furthermore the calculation of the intersection indicator imposes no extra cost since we can reuse the calculated value in the calculation of intersection points. Vielleicht noch dazu dass beim betrachten eines modells the bspheres wie ein matel um das modell sind und diese Fälle dann ausgeschlossen werden.

4. **Calculate intersection points** This operation is somewhat more involved in GA than in LA. In LA you can simply calculate the ray parameter t1 and t2. Using the parameters, the origin and direction of the ray one can easily calculate the intersection points. In GA the result of the intersection is a point pair which includes both of the intersection points. We have to dissect the point pair to get the intersection points. After dissection we have to normalize the intersection points so that the following tests work in the right way.

5. **Test intersection points against bounding sphere** We have to introduce this test before we can expect the intersection points to be feasible points of our model. There are cases where a ray intersects both the bounding sphere and the fitting sphere but only one of the possible two intersection points or none of them is feasible concerning our model description. In figure 2 you can see three of the cases depicted in 2D that can occure when a ray intersects both spheres. The top case is the usual case where both of

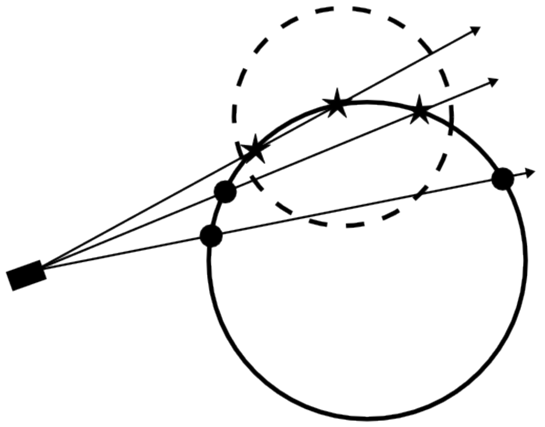


Figure 2: Three cases of the intersection of a ray with both the bounding sphere and the fitted surfel. The surfel is depicted in black while the bounding spheres outline is stippled. There are three feasible intersection points depiected by the stars and three infeasible intersection points represented by the black filled circles

the intersection points are feasible and we take the nearest one for further calculations. In the middle case only one of the intersection points is inside the bounding sphere. The lower case shows an example of the intersection of both spheres where none of the intersection points is feasible for our model.

6. **Test intersection points to be the nearest points** We use the inner product of Geometric Algebra to get the distance measure between an actually found intersection point and a point from a previous iteration of the algorithm.

7. **Repeat 1. - 6. for other candidate fits**

After finding the nearest intersection point for the ray we shade the pixel using the Phong lighting model. To calculate the diffuse part of the shading the Phong model needs the normal at the intersection point. Calculating the normal is the first time we have to branch depending on the type of our surfel. The normal of a plane surfel can be read directly from the GA coefficients. The normal of a sphere surfel can be calculated from the center of the sphere to the intersection point.

4 MINIMIZING OPERATIONS

One subgoal of this investigation was to compare our GA raytracing approach to existing solutions which are based on Linear Algebra and their amount of primitive operations like additions and multiplications. We looked at the following raytracing subtasks:

- Determining whether a ray intersects an object in the scene, in our case spheres
- Calculating the intersection point of a ray and an object
- Finding the surface normal at the intersection point
- Calculating the reflection vector which is needed for lighting and recursive raytracing

The number of operations for the case of LA was taken from [22] where a GPU raytracing method based on quadrics is introduced. Our algorithm was developed and tested with CluCalc [18]. To analyse and increase the performance of our algorithm we used the tool Gaalop [11] to symbolically optimize our GA formulas. The output of Gaalop was then searched for further potential of optimization with the intend of reducing the total number of multiplications and additions/substractions. This way differs from the approach of optimization in [6] where a GA raytracer is implemented on a CPU with the help of Gaigen2 [5]. Gaigen2 increases the performance of GA algorithms by using specialized objects instead of standard 32 entry multivectors. This leads to the advantage that only those entries are considered in calculations which really belong to an object. For example a sphere will always contain only non-zero coefficients for e_1, e_2, e_3, e_{inf} and e_0. In general we used three ways to reach our goal of less primitive operations:

1. We searched for constant values of variables which could be excluded from the calculation. Especially constant values of zeroes and ones lead to simplifications.
2. Gaalop sometimes calculates coefficients which don't belong to the object. These calculations can be removed completely.
3. Sometimes the same multiplications appear in more than one coefficient or more than one time in the same coefficient. These parts were factored out. They can be precalculated in a previous step and the result is used in the computation of the coefficients.

All optimizations were done under the point of view that the algorithm should run on a parallel platform and especially on GPUs and FPGAs. Point 3 of the list above could be important for implementing the algorithm on a FPGA, because it implies a pipeline architecture where the result is calculated in some single steps. This type of architecture can be computed on a FPGA in an efficient way if it is assured that the pipeline can be filled constantly with new data. This requirement is fullfilled in raytracing applications because of the high number of pixels for which the raytracing procedure must be executed. Another intend during the developement of the algorithm was to avoid the use of square roots and divisions because they cause lot of computational effort. In the LA case most divisions and roots are caused by the normalization operation. In GA we are in most cases able to use unnormalized objects because a scaled multivector represents the same geometric object in the conformal space. To demonstrate our approach we describe in detail the inspection of the ray-sphere intersection and reflections in the following two sections.

4.1 Ray-Sphere Intersection

In the GA case we have two objects. The sphere S and the ray R, which is represented by a line. The surface normal is represented by a line, too. In the following $*$ denotes the geometric product of two entities, while the inner product is represented by the . operator. S and R are intersected through the outer product $S \wedge R$. The result of this operation is the point pair Pp. We have to extract the point P from Pp which is nearest to the eyepoint. For this extraction the following formula is used:

$$P = (\sqrt{Pp.Pp} \pm Pp) * (einf.Pp)$$

This is a variation of the extraction formula from [12, p. 74, 6.11] where the division through $einf.PP$ is replaced by a geometric product. This represents the same object in GA like described above.

The inner product $Pp.Pp$ indicates whether the ray intersects the sphere or not. If it is positive there exists an intersection. The calculation of this value consists of a long sum of products and can't take advantage of parallelism. In LA it is necessary to compute the discriminant of a quadratic equation which leads to 8 additions and 20 mutliplications. In GA we need 14 additions and 22 multiplications. So the effort for the decision of intersection is comparable in both algebras. Considering all subtasks, the inspection of [22] and counting of operations lead to the amount of calculations summarized in table 1.

	GA	LA
additions/substractions	29	12
multiplications	42	23
divsions	0	1
square roots	1	1

Table 1: operations for intersection

So in general GA needs two times more operations than LA. But a point consists of five non-zero coefficients which can be calculated in parallel. In LA there are only three entries in the result vector which can be computed simultaneously. As a result it seems possible that the GA computations can be performened in the same time as those from LA on a parallel hardware.

4.2 Reflections

The field of reflections was analysed with the most effort of the four subtasks. First we looked at reflections in the 5D conformal space. To reflect an incoming ray on a sphere we construct a temporary plane PL through the intersection point P in the direction of the surface normal N in P. With the help of two geometric products we can calculate the reflected ray R_{ref} by the formula:

$$R_{ref} = -Pl * R * PL$$

The resulting C-code created by Gaalop contained over 8 times more primitive operations than the LA solution. By hand optimizations lead to a solution which is still between 5 and 6 times larger than the existing LA solution with the help of the formula. Further optimization seems not to be possible. Another approach was to use a rotation around the normal. This is possible because our normal is represented as a line and not as a direction vector like in LA. But this way also leads to results comparable to the temporary plane solution. This is due to the characteristic of 5D GA that all calculations are done free in space and not like in LA related to the origin what leads to a clearly higher computational effort. So we analyzed the reflection in 3D GA. Our investigation was based on [23, p. 108f] and took into account two different variants of the GA description of reflection. The first one is the equivilant to our 5D solution and taken from [23, p. 109, 8.28]. The refleced ray is calculated through two geometric products of the ray R and the normal N of the plane PL.

$$R_{ref} = -N * R * N$$

This leads to C-code with 3 times higher operation count than LA. After by hand optimization the operation count can be reduced to the same amount as in LA. But to do this we had to presume that the constructed normal has unit length, which has to be achieved through cost intensive normalization. However because of our use of the Phong model we need the normalized direction of the reflected ray anyway, so that this is not drawback.

	GA 5D	GA 3D	LA
additions/substractions	25	5	5
multiplications	37	7	7

Table 2: operations for reflection

So the GA and LA solution have the same amount of operations in 3D. Like in LA the result vector contains 3 elements which can be computed in parallel.

5 IMPLEMENTATION DETAILS

In our fitting process we find the neighbors of the fitting points by using a knn-search of the ANN library [15]. We use the newmat library [2] to calculate the eigenvectors of the 5x5 matrix computed from the point neighborhood.

For our final implementation we use the OpenCL environment together with an AMD Radeon HD4850 graphics card to raytrace views of our scenes. We create a 256 by 256 array of threads which is the maximum for our hardware in the OpenCL environment. The array of threads is slided across the image domain so that every pixel is covered once by one thread. Every thread of the thread array calculates the raytracing algorithm for the covered pixel independent of the other threads.

We derive the rays by calculating a GA line using the origin of the camera and a point on the image plane. The point on the image plane is interpolated bilinearly in euclidean space from the 3D coordinates of the image planes corners depending on the pixel position and the image resolution.

To speed up the calculation of intersections we use a modified kd-tree. When we split a node of the kd-tree in the building process we can not cut surfels that cover the splitting plane of the kd-tree node. We decided to enlarge the axis aligned bounding boxes (AABB) of the resulting child nodes so that every surfel is enclosed completely in exactly one of the child nodes. We assign the surfels to the child node that is covered by most of

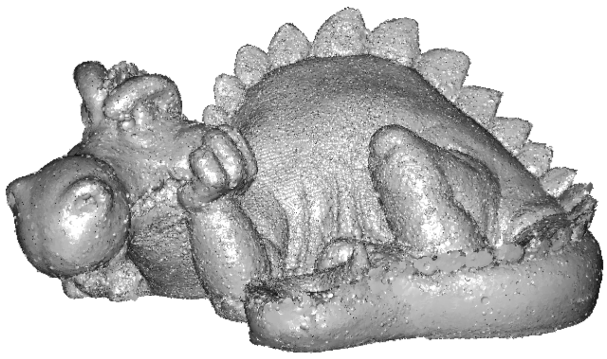


Figure 3: Phlegmatic Dragon point cloud rendered at 2.6 fps on an AMD HD4850 GPU. The viewport size is 640 by 480. The model consists of 166162 surfels.

the surfels AABB volume. A result of our kd-tree building process is that several of the trees node AABBs will intersect each other which does not happen in a real kd-tree. Furthermore we have to use a larger memory footprint than is necessary for the usual kd-tree because we save six coordinates for two corner points of the nodes AABBs instead of only saving one coordinate for the splitting plane and its direction. To traverse the kd-tree we use a stack in OpenCLs shared memory. On our hardware shared memory is emulated in global memory. A result of this is that on our hardware the large memory footprint is not such a big problem. Instead of reading the AABB information from the stack like one would do it with the usual kd-tree we read the coordinates from our kd-tree data structure. Both data sources are in global memory.

6 RESULTS

We chose two different architectures to benchmark our algorithms. We implemented a CPU version to measure the impact of choosing one of the algebras for the raytracing application directly. Our second architecture are AMD 45xx series GPUs. The AMD GPUs can be seen as a parallel processor for one invocation of the raytracing algorithm for one pixel. For further details we refer to section 6.2.

6.1 Raytracing on CPUs

Our CPU implementation of the raytracing algortihm is written in plain C/C++. We do not use any vector extensions like SSE since we are interested in the performance of LA and GA based on their mathematical effort. To speed the algorithm invocation up we split the calculations for the pixel array of the final image up among multiple CPU cores with OpenMP pragmas.

6.2 Raytracing on GPUs

We decided to use the AMD Stream Technology for our benchmarks on GPUs to get two advantages. First the AMD processing elements that compute a single thread of our GPU raytracing kernel have a 5-way VLIW Design (very long instruction word). The processing elements can compute up to five floating point additions or multiplications simultaneously which make up most of the calculations of our GA raytracing algorithm. Second with AMD Stream Kernel Analyzer we can disassemble the compiled code for the GPU. Let's look at the example of calculating the reflection in Linear Algebra. In listing 1 you can see the according kernel written in the Brook+ language.

```
kernel void reflection(float4 incident_ray<>,
  float4 normal<>, out float4 reflected_ray<>){
      float factor;
      float4 ret;
      factor = 2.f*(incident_ray.x*normal.x +
      incident_ray.y*normal.y+
      incident_ray.z*normal.z);
      ret.x = incident_ray.x − normal.x * factor;
      ret.y = incident_ray.y − normal.y * factor;
      ret.z = incident_ray.z − normal.z * factor;
      ret.w = 0.f;
      reflected_ray = ret;
}
```

Listing 1: Brook+ kernel that calculates the reflection in Linear Algebra

The disassembly of the compiled code can be seen in listing 2. There are five instructions denoted by the numbers 2 to 6. The characters x, y, z, w and t show which of the five ALUs are active in one instruction. In instruction 2 there are four active ALUs of which three calculate a multiplication while ALU t issues a move operation. In contrast instruction 3 has only one active ALU.

```
2  x: MUL_e*2 T0.x, R1.z, R0.z
2  z: MUL_e*2 ____, R1.y, R0.y
2  w: MUL_e*2 ____, R1.x, R0.x
2  t: MOV R2.w, 0.0f
3  y: ADD ____, PV2.w, PV2.z
4  w: ADD ____, PV3.y, T0.x
5  x: MUL_e ____, R0.z, PV4.w
5  y: MUL_e ____, R0.y, PV4.w
5  z: MUL_e ____, R0.x, PV4.w
6  x: ADD R2.x, R1.x, −PV5.z
6  y: ADD R2.y, R1.y, −PV5.y
6  z: ADD R2.z, R1.z, −PV5.x
```

Listing 2: The computational part of the reflection dissassembly

With the advantages of parallel execution of operations inside a thread and the possibility to dissassmble the compiled code we can directly measure the impact of the parallel nature of GA multivector calculations compared to an algorithm in Linear Algebra. The measurement is possible both in terms of instruction count by using the disassemblies of the according kernels and in terms of execution time during a benchmark. The results in table 3 show that our theoretical considerations which lead to the conclusion that the GA algorithm has advantages on parallel architectures compared to LA were right.

	LA inst	GA inst	GA/LA inst	GA/LA op
II	9	12	1.33	1.29
IP	13	17	1.3	2.0
RF	5	12	2.1	5.2

Table 3: GA/LA instructions and operations in comparision for Intersection Indicator (II), Intersection Point (IP) and Reflection (RF)

The table compares the instruction count of both algebras (LA inst and GA inst). Furthermore the ratio for the instruction count (GA/LA inst) and the observed ratio for the operation count (GA/LA op), which was derived in section 4, between them is shown. It is obvious that the instruction ratio for the intersection point and especially for the reflection vector is significantly smaller than the operation ratio. So the GA multivectors can profit from AMDs architecture that puts up to five operations into one instruction. The value for the intersection indicator doesn't change because its computation can't be parallelized like shown in section 4.

6.3 Performance

We use artificial scenes and real world point clouds for our benchmarks. The artificial scenes are designed to show the impact of different stages in our raytracing algorithm. For the intersection indicator we create a scene consisting of 400 spheres. The spheres are placed in a screen aligned 2D grid with a distance of their center equal to 1.4 times their radius. As a result every ray through a pixel of the image intersects at most two spheres. The scene designed for the intersection consists of 100 screen filling spheres that are placed one behind the other to get a high depth complexity. A single screen filling sphere is used to benchmark the impact of the reflection calculation for shading. The real world scenes are covered by the Egea model and a reduced version of the chameleon point cloud with around 4500 points.

The CPU implementation is consistent with our theoretical observations in section 4. Table 4 shows that the GA needs more time to render the same scene than LA. The difference is not that high like in theory because there is a mixture of different parts of the algorithm even if the scene is designed to show the impact

	II	IP	RF	CH	EG
GA	2320	2190	270	21700	39089
LA	2000	1901	140	19580	34931
GA/LA	1.2	1.2	1.9	1.1	1.1

Table 4: Timings in milliseconds for different scenes on an AMD Athlon 5600+ dual core at 2.8 GHz. Intersection Indicator (II), Intersection Point (IP) and Reflection (RF) are artificial scenes to benchmark the different parts of the raytracing algorithm. The real world scenes are Chameleon (CH) and Egea (EG). The viewport size is 320 by 240

of one special part of the raytracing algorithm. Furthermore the timings show the need for a spacial data structure to render real world scenes.

	II	IP	RF	Chameleon	Egea
GA	52	49	14	770	1402
LA	68	62	32	810	1322
GA/LA	0.76	0.79	0.43	0.95	1.06

Table 5: Timings in milliseconds for different scenes on an AMD HD4850 GPU of the brute force approach. Intersection Indicator (II), Intersection Point (IP) and Reflection (RF) are artificial scenes to benchmark the different parts of the raytracing algorithm. The viewport size is 640 by 480

Looking at the results of the GPU brute force implementation presented in table 5 we were somehow surprised. The GA algorithm does not only perform comparable to the LA implementation but is in some cases even faster. An investigation of the dissasemblies of both kernels shows a 5,26% increase in ALU instructions and a 28% increase in control flow instructions for the LA implementation compared to GA. The increase in control flow instructions seems to affect only the artificial scenes with its low object count. The real world scenes are rendered with nearly equal speed.

	Bunny	Max Planck	Dragon
surfel count	19336	96208	166162
time (ms)	160	220	380

Table 6: Performance of the final implementation using OpenCL on an AMD HD4850 GPU. The surfel models were computed from the Standford Bunny, Max Planck and original version of the Phlegmatic Dragon point clouds. The viewport size is 640 by 480

The performance of our final OpenCL implementation presented in table 6 shows that we get interactive frame rates for small and medium sized scenes.

7 CONCLUSION

We were able to show that it is possible to optimize GA code in a way, that its amount of primitive operations is comparable to that of LA solutions, with a little drawback for GA. In some cases we had to do some handwork to improve the results obtained by Gaalop even further. In the field of reflections it does not seem to be possible to reach a comparable count of operations in the case of conformal space. To clearly increase the performance we have to use the 3D case with the drawback to transform between spaces and to loose some part of the elegance and compactness of GA. So there exists always the task of finding a trade-off between performance and elegance/compactness depending on the kind of application which is developed.

Furthermore by using parrallel architectures one can reach comparable speed for algorithm in GA and LA even without additional handwork. Though the difference in speed for both algebras does not only depend on mathematical effort but on other factors like the amount of control flow instructions.

8 FUTURE WORK

We aim to implement our GA raytracing procedure on a FPGA. Important questions to answer will be how to partition the algorithm between FPGA and a CPU, what spacial data structure to use and the relation between pipeline and parallel architecture. Existing raytracing procedures and GA co-processors on FPGAs reached the capacity of the hardware very fast. So the challenge will be to combine both tasks on a single component.

We also want to compare our introduced surfel model to other surface representations to render point clouds. We aim to improve the visual quality of our model by interpolating neighboring surfels to get a smooth surface without discontinuities. To enhance the rendering speed, which is about an order of magnitude below other algorithms to render point clouds, we will investigate which spatial data structure is suited better to our model than the presented kd-tree.

REFERENCES

[1] AMD. The AMD Stream Technology home page. HTML document http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx, 2009.

[2] Robert Davies. The Newmat home page. HTML document http://www.robertnz.net/nm_intro.htm, 2009.

[3] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufman, 2007.

[4] Joshua Fender. A high-speed ray tracing engine built on a field-programmable system. In *Proc. Int. conf. on Field-Programmable Technology, IEEE*, pages 188–195, 2003.

[5] D. Fontijne, T. Bouma, and L. Dorst. Gaigen 2: A geometric algebra implementation generator. http://staff.science.uva.nl/ fontijne/gaigen2.html.

[6] Daniel Fontijne. *Efficient Implementation of Geometric Algebra*. PhD thesis, University of Amsterdam, 2007.

[7] S. Franchini, A. Gentile, M. Grimaudo, C.A. Hung, S. Impastato, F. Sorbello, G. Vassallo, and S. Vitabile. A sliced coprocessor for native Clifford algebra operations. In *Euromico Conference on Digital System Design, Architectures, Methods and Tools (DSD)*, 2007.

[8] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 23, New York, NY, USA, 2007. ACM.

[9] D. Hildenbrand. Geometric computing in computer graphics using conformal geometric algebra. *Computers & Graphics*, 29(5):802–810, 2005.

[10] D. Hildenbrand, H. Lange, Florian Stock, and Andreas Koch. Efficient inverse kinematics algorithm based on conformal geometric algebra using reconfigurable hardware. In *GRAPP conference Madeira*, 2008.

[11] D. Hildenbrand and Joachim Pitt. The Gaalop home page. HTML document http://www.gaalop.de, 2008.

[12] Dietmar Hildenbrand. *Geometric Computing in Computer Graphics and Robotics using Conformal Geometric Algebra*. PhD thesis, Darmstadt University of Technology, 2006.

[13] Intel. The Ct: C for Throughput Computing home page. HTML document http://techresearch.intel.com/articles/Tera-Scale/1514.htm, 2009.

[14] Takashi Kanai, Yutaka Ohtake, Hiroaki Kawata, and Kiwamu Kase. Gpu-based rendering of sparse low-degree implicit surfaces. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 165–171, New York, NY, USA, 2006. ACM Press.

[15] David M. Mount and Sunil Arya. The ANN home page. HTML document http://www.cs.umd.edu/ mount/ANN/, 2009.

[16] NVIDIA. The CUDA home page. HTML document http://www.nvidia.com/object/cuda_home.html, 2009.

[17] Yutaka Ohtake, Alexander Belyaev, and Marc Alexa. Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing.

[18] C. Perwass. The CLU home page. HTML document http://www.clucalc.info, 2008.

[19] C. Perwass, C. Gebken, and G. Sommer. Implementation of a Clifford algebra co-processor design on a field programmable gate array. In R. 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.

[20] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, pages 343–352, July 2000.

[21] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. Saarcor – a hardware architecture for ray tracing. In *Proceedings of the conference on Graphics Hardware 2002*, pages 27–36. Saarland University, Eurographics Association, 2002. available at http://www.openrt.de.

[22] C. Stoll, S. Gumhold, and H.-P. Seidel. Incremental raycasting of piecewise quadratic surfaces on the gpu. *Symposium on Interactive Ray Tracing*, 0:141–150, 2006.

[23] John Vince. *Geometric algebra: An algebraic system for computer games and animation.* London: Springer. xviii, 195 p, 2009.

Realtime KLT Feature Point Tracking for High Definition Video

Hannes Fassold[1]
hannes.fassold@joanneum.at

Jakub Rosner[2]
jakub.rosner@joanneum.at

Peter Schallauer[1]
peter.schallauer@joanneum.at

Werner Bailer[1]
werner.bailer@joanneum.at

ABSTRACT

Automatic detection and tracking of feature points is an important part of many computer vision methods. A widely used method is the KLT tracker proposed by Kanade, Lucas and Tomasi. This paper reports work done on porting the KLT tracker to the GPU, using the CUDA technology by NVIDIA. For the feature point detection, we propose to do all steps of the detection process, except the final one (enforcing a minimum distance between feature points), on the GPU. The feature point tracking is done on a multi-resolution image representation to allow tracking of large motion. Each feature point is calculated in parallel on the GPU. We compare the CUDA implementation with the corresponding OpenCV (using SSE and OpenMP) routines in terms of quality and speed, noticing a significant speedup of up to factor 10. Some additional experiments are done regarding the influence of different parameterization on the runtime. Our GPU implementation achieves realtime (> 25 fps) performance for High Definition (HD) video sequences, successfully tracking several thousands of points. In summary, the GPU implementation achieves a significant speedup compared with an optimized CPU implementation and allows the analysis of high resolution video sequences in realtime.

Keywords

KLT, feature point tracking, Lucas Kanade, corner detection, optical flow, motion estimation, GPU, CUDA

1. INTRODUCTION

The automatic detection and tracking of (typically corner-like) feature points throughout an image sequence is a necessary prerequisite for many algorithms in computer vision. The gathered information about the feature points and their motion can be used subsequently for pose estimation, camera self-calibration [Koc99] and for tracking various kinds of objects like people and vehicles [Lyp07][Kan06]. One of the most popular methods for feature point tracking is the KLT algorithm which was introduced by Lucas and Kanade [Luc81] and later extended in the works of Tomasi and Kanade [Tom91] and Shi and Tomasi [Shi94]. The KLT algorithm automatically detects a sparse set of feature points which have sufficient texture to track them reliable. Afterwards, detected points are tracked by estimating for each point the translation, which minimizes the SSD dissimilarity between windows centered at the current feature point position and the translated position.

Despite being more than 20 years old, the KLT algorithm is still widely used, as it operates in a fully automatic way and its performance in terms of feature point quality and runtime is competitive compared with other methods. A problem occurs, when using the KLT algorithm in realtime applications (e.g. in surveillance), where strict runtime requirements must be fulfilled. Typically cameras deliver 25 – 30 images per second, so the runtime of the algorithm for one image may not exceed 33 - 40 milliseconds.

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

[2] Silesian University of Technology, Faculty of Automatic Control and Robotics, Ulica Academicka 2, 44-100 Gliwice, Poland

Current implementations of the KLT algorithm (e.g. the OpenCV[3] routine) achieve this only when the image resolution is not higher than Standard Definition (720x576) and the number of feature points is a couple of hundreds at most. If the image resolution is higher (e.g. for HD video) and more points are to be tracked, one has to look for alternatives.

Because of its tremendous computational capability Graphic Processing Units (GPUs) gain significant importance for computer vision. In this document we describe work done on porting the KLT algorithm to the GPU using CUDA. CUDA[4] stands for Compute Unified Device Architecture and is a C-like GPU programming environment introduced by NVIDIA. We first give an introduction to GPU programming with a focus on CUDA (section 2) and discuss previous work done on implementing the KLT algorithm for the GPU (section 3). In section 4, an overview of the general KLT algorithm is given and section 5 discusses its implementation for the GPU. Finally, section 6 compares the GPU implementation with the reference CPU implementation in terms of speed and quality.

2. GPU PROGRAMMING & CUDA

In the last few years, GPUs have evolved from specialized devices for accelerating 3D graphics to powerful coprocessors, which can be used for general purpose GPU programming. The processing power of GPUs (measured as number of floating-point operations per second) is nearly doubling every year and exceeds modern CPUs processing capabilities by far. Moreover, while a couple of years ago GPU developers had to adapt their algorithms to fit into a special purpose computer-graphics oriented render pipeline, the advent of general purpose GPU programming languages like Brook[5], CUDA or OpenCL[6] brought much more flexibility into the field of GPU programming. In the following, we will focus on CUDA, which is currently the most mature of these and can be run on all current NVIDIA GPUs starting with the Geforce 8 series.

CUDA GPU Architecture

The following properties are characteristic for a CUDA-capable GPU:

- Manycore architecture (e.g. Geforce 280GTX has 30 multiprocessors, corresponding to 240 processing cores),
- A very fast thread management (done in hardware), which allows switching between different threads with virtually no overhead,
- Random access device memory (*global memory*) which can be accessed by all threads, but has a high latency,
- A very fast read-write cache called *shared memory* (16 KB per multiprocessor), which has to be managed by the algorithm developer,
- Other important memory types like *texture memory* (read-only, cached, offers bilinear interpolation) and *constant memory* (read-only, cached).

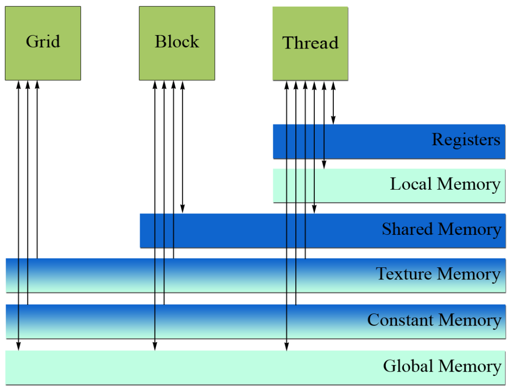


Figure 1: Different memory types of a CUDA-capable GPU. Blue = on-chip, yellow = off-chip, shaded = off-chip, but cached. The arrows indicate the allowed access type (read-only, read-write).

CUDA Programming Model

A CUDA program is typically composed of a control routine which calls a couple of CUDA *kernels.* A *kernel* can be compared to a C function, but is executed on the GPU in parallel by a large number of threads in a SIMT (single instruction, multiple threads) fashion. Each *thread* is identified by its unique *thread id*. Groups of 32 consecutive threads are organized into *warps* with *half-warps* as their first or second halves. Furthermore, sets of up to 512 consecutive threads are grouped into *thread blocks*, which then form a *grid*.

Synchronization among different thread blocks can only be achieved after the whole kernel has completed (*global synchronization*). Depending on the resources (registers, shared memory) a thread block uses, one or more of them are assigned to a multiprocessor to be executed simultaneously. After having finished, new thread blocks are assigned to the multiprocessor, until the whole grid has been completed. The order in which thread blocks are executed is not defined and depends on the number of multiprocessors of the GPU.

[3] http://sourceforge.net/projects/opencv/

[4] http://www.nvidia.com/object/cuda_home.html

[5] http://graphics.stanford.edu/projects/brookgpu/

[6] http://www.khronos.org/opencl/

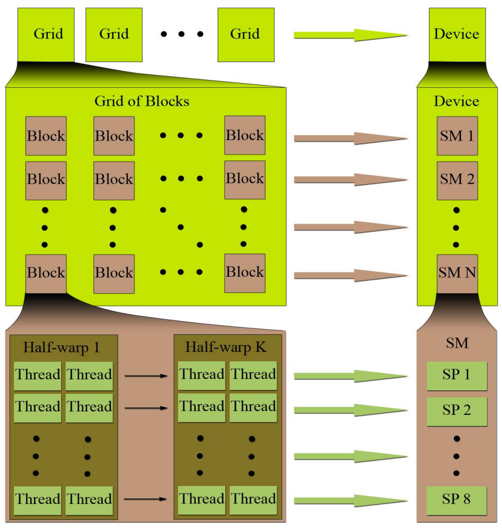


Figure 2: CUDA assigns grids to a device (GPU), thread blocks to its Streaming Multiprocessors (SM) and threads to Scalar Processors (SP).

Note that in image processing algorithms, typically one thread computes one pixel. A thread block is typically corresponding to a small tile of the image. All thread blocks corresponding to the whole image then form the grid.

CUDA Porting Guidelines

In the following section, some general guidelines are given how to port an algorithm efficiently to CUDA. First (and most important), it must be able to split the algorithm into a large number (at least hundreds) of loosely coupled threads which run in parallel on the GPU. One often has to rethink the whole algorithm or parts of it to be able to fulfill this requirement.

A very common way in writing a CUDA kernel is to divide it into three parts separated by a synchronization barrier. In the first part all the data essential for computations inside a given thread block is loaded from global memory to the very fast shared memory. It is essential to use a synchronization barrier after the loading stage to ensure that all the data has been loaded before proceeding to the next step. In the next part, the processing stage, the shared memory data is used in a way that depends on the purpose of the given algorithm - e.g. a convolution, interpolation, summation etc. The results are then stored in a temporary buffer residing also in shared memory. Another synchronization barrier has to be set to ensure completeness of the processing stage. The last part is usually the shortest one and simply writes the temporary buffer back into the proper location in global memory.

An obvious and troublesome problem for the programmer is the very high level of parallelism in kernel's execution. There can be tens of thousands of threads running concurrently on a single GPU. When two threads try to increment, compare or change in some other way the value at the same memory address simultaneously, only one of them will be likely to succeed. In that case it is strongly recommended to think of another, sometimes completely different way to implement the given algorithm to allow parallel execution. This problem can be very hard and in some cases the only way to solve it is to use atomic functions. Atomic functions will be completely serialized and for this reason can significantly decrease the overall performance.

Branches ('if-then-else', 'while') within the threads of a half-warp should be reduced to a minimum as they lead to divergent execution of threads, and are sometimes serialized. Memory transfers between CPU memory and GPU memory should also be reduced to minimum as they are very costly. The same applies (to a lesser extent) to allocations and deallocations of large memory buffers.

The available memory types as shown in Figure 1 should be understood and used properly to achieve an optimal implementation. Especially the usage of shared memory to cache a small part of the high-latency global memory is important. Furthermore for optimal performance in accessing global memory, threads within a half-warp should access memory locations of the same memory segment (*coalescing rule*). It is possible to hide a large part of memory latency in arithmetic computations by executing as many threads simultaneously as possible (the limit is 1024 per multiprocessor for the latest GPUs). Furthermore, for read-only data the usage of texture and constant memory is often helpful as they are cached.

Interactions between threads should be restricted to threads of the same thread block and done using shared memory. Shared memory is especially useful in cases when those threads use data that mutually overlaps and is generally as fast as registers, as it resides on chip, unless *bank conflicts* occur. To avoid them, threads from the same half-warp have to read memory addresses with a step size which is dividable by 4 bytes and not dividable by 8 bytes.

Register usage of a single thread block should be minimized, otherwise the number of thread blocks that can be run concurrently by a single multiprocessor may be reduced.

Shared memory, which is mainly used for communication between threads of the same block, is a very powerful ally in kernel optimization. Regarding the KLT algorithm implementation, it is used in almost every CUDA kernel and greatly improves the performance. Shared memory can also be successfully used to store small per-thread arrays, as otherwise they would be allocated by the compiler outside the chip in the *local memory*, which has the same latency as the global memory.

For some special functions like square-root(), sine() and cosine() there exist significantly faster variants with lower precision. Although available since the NVIDIA G200 GPU series, the usage of *double* precision values should be avoided as it is significantly slower than single precision ones.

Note that in [Che08] a study has been done about the effectiveness of CUDA when porting different kind of algorithms (combinatorial logic, dynamic programming, data mining etc.) to the GPU. They report speedups ranging from moderate 2.9 times for dynamic programming (which is hard to parallelize) up to 72 times for k-mean clustering.

3. RELATED WORK

There has been done some previous work on porting the KLT tracker to the GPU. In [Sin06] OpenGL and the Cg[7] shader language are used for the GPU implementation. A fixed number of iterations is done for each feature point to avoid conditional statements. Detection of new feature points is done only every fifth frame to save computation time. In [Zac08] the KLT tracker is also implemented using Cg. Their KLT tracker compensates for varying camera gain by estimating it as a global multiplicative constant. Another Cg KLT implementation is described in [Ohm08]. They propose a modified variant of the feature detection process to circumvent some hard parallelizable parts of it. Note that to our knowledge, no CUDA KLT implementation has been reported so far in published works.

4. KLT ALGORITHM

The KLT algorithm can be divided into two main parts. During the detection process, salient feature points are found and added to the already existing ones. Afterwards, in the tracking process for each feature point its corresponding motion vector is calculated. In the following, we describe each part of it in more detail. The algorithm follows the standard scheme for the KLT algorithm as was proposed in the works of [Luc81][Tom91][Shi94].

Note that in the following Greek letters denote scalars, lowercase letters denote column vectors and uppercase letters denote matrices. We denote *I* as the current image and *J* as the immediately next image in the sequence. We write $\nabla I = \partial I / \partial(x,y)$ as the spatial image gradient of *I*, which is typically done with the Sobel or Sharr operator for robustness. Also we define as *W*(*p*) a small rectangular region centered at a given point *p*. Typically *W*(*p*) will be a 5 x 5 or 7 x 7 pixel neighborhood. As the tracking is done with sub-pixel precision, *p* will have non-integer coordinates. Its neighbors are then calculated using bilinear interpolation.

Feature Point Detection

The task here is to detect new feature points in a given image *I* and add them to the already existing feature points. In order to track feature points reliably, their pixel neighborhood should by richly structured. As a measure of 'structuredness' of the neighborhood of a pixel *p*, one can define the structure matrix *G*:

$$G = \sum_{x \in W(p)} \nabla I(x) \cdot \nabla I(x)^T$$

Its eigenvalues λ_1, λ_2 (which are guaranteed to be ≥ 0 as the matrix is positive-semidefinite) deliver useful information about the neighborhood region *W*. If *W* is completely homogenous, then $\lambda_1 = \lambda_2 = 0$. In contrast, $\lambda_1 > 0$, $\lambda_2 = 0$ indicates that W contains an edge and $\lambda_1 > 0$, $\lambda_2 > 0$ indicates a corner. The smaller eigenvalue $\lambda = \min(\lambda_1, \lambda_2)$ can now be used as a measure of the cornerness of *W*, where larger values means stronger corners.

The feature detection is now composed of the following steps:

1. Calculate structure matrix *G* and cornerness λ for each pixel in the image *I*.
2. Calculate the maximum cornerness λ_{max} occurring in the image.
3. Keep all pixels that have a cornerness λ larger than a certain percentage (5% - 10%) of λ_{max}.
4. Do a non-maxima suppression within the 3 x 3 pixel neighborhood of the remaining points to keep only the local maxima.
5. From the remaining points, add as many new points to the already existing points as needed, starting with the points with the highest cornerness values. To avoid points concentrated in some area of the image, newly added points must have a specific minimum distance (e.g. 5 or 10 pixels) to the already existing points as well as to other newly added points (*Minimum-Distance-Enforcement*).

Feature Point Tracking

In the tracking step, we want to calculate for each feature point *p* in image *I* its corresponding motion vector *v* so that its tracked position in image *J* is *p* + *v*.

As 'goodness' criterion of *v* we take the SSD error function $\varepsilon(v) = \sum_{x \in W(p)} (J(x+v) - I(x))^2$. It measures

[7] http://developer.nvidia.com/object/cg_toolkit.html

```
1. Set initial motion vector $v_1 = (0,0)^T$
2. Spatial image gradient $\nabla I = \partial I / \partial(x,y)$
3. Calc. structure matrix $G = \sum_{x \in W(p)} \nabla I(x) \cdot \nabla I(x)^T$
4. for $k$ = 1 to $maxIter$
    a) Image difference $\eta(x) = I(x) - J(x + v^k)$
    b) Calc. mismatch vector $b = \sum_{x \in W(p)} \eta(x) \cdot \nabla I(x)$
    c) Calc. updated motion $v_{k+1} = v_k + G^{-1} b$
    d) if $\| v_{k+1} - v_k \| < eps$ then stop (converged)
5. Report final motion vector $v$
```

Table 1: Pseudo-code of the calculation of the motion vector *v* for a given feature point *p*. W(*p*) is a window centered at *p*. Typically the window size is set to 5 x 5 pixel, *maxIter* to 10 and *eps* to 0.03 pixel.

the image intensity deviation between a neighborhood of the feature point position in *I* and its potential position in *J* and should be zero in the ideal case. Setting the first derivative of $\varepsilon(v)$ to zero and approximating *J*(*x* + *v*) by its first order Taylor expansion around *v* = 0 results in a better estimate v_1. By repeating this multiple times, we obtain an iterative update scheme for *v* which is summarized in Table 1.

Due to the Taylor expansion around zero the given scheme is only valid for small motion vectors *v*. In order to allow tracking of large motions of feature points, which is quite common, we generate an image pyramid and apply the scheme for all points in each pyramid level. We are doing this from coarse pyramid level to fine one, using the result of the previous pyramid level as initial guess for the next one.

5. CUDA IMPLEMENTATION

In this section we will discuss issues which are specific for the CUDA implementation of the KLT tracker.

The very first thing that has to be done before any GPU kernel can be run is to allocate a GPU memory buffer and transfer the essential data from CPU memory to it. In order to save unnecessary allocations and deallocations of GPU memory, before processing the first image of the sequence a context object is created which holds all the necessary memory resources (for the image pyramids etc.) for the KLT algorithm. It is reused during processing of the image sequence and deleted after the processing is finished.

Feature Point Detection Implementation

The algorithm which does the feature point detection has been divided into separate steps, as explained in section 4, each being computed by one or more kernels. For optimization purposes the operations were assigned to kernels in a way that minimizes the overall number of kernels and by that read-write operations in global memory space, as those are particularly costly.

The first step of the algorithm has been divided into three different kernels, first computing the gradients for each pixel and the next two summing them up in window *W* in order to get the *G* matrix and the cornerness λ.

The second step, which determines the maximum cornerness, is a good example of an operation that seems to be conceptually very simple, but is complicated to implement efficiently for a massively parallel architecture. In this case it involves a lot of read-write hazards, when many threads want to compare and modify a single value simultaneously. A solution to this problem is to create a *reduction tree*, in which each thread determines the maximum of a couple of values and stores it at a different address. A highly optimized version of this reduction algorithm (along with some other useful algorithms for compaction, sorting etc.) can be found in the CUDA performance primitives library[8] (CUDPP) and was used by us for calculating the maximum value.

In steps 3 and 4 we mark features, that do not meet their respective conditions, as invalid, since removing them in each step separately would not only be complicated on GPU, but also inefficient.

Before doing step 5, the potential feature points have to be transferred back to the CPU memory. Considering how costly such transfers are, the feature points are compacted before that to remove the ones, which were marked as invalid. Once again, we use the CUDPP library for this purpose.

Step 5 of the feature point detection algorithm (the enforcement of a certain minimum distance between feature points) is very hard to parallelize and inefficient to run on the GPU. This fact forces us to do it on the CPU. Note that the standard algorithm as implemented in the OpenCV library is only efficient for a small number of features (less than 1000) as its computational complexity increase quadratically. For efficiently handling larger numbers of features we have implemented an alternative algorithm, whose complexity increases linearly with the number of features. It requires an additional mask image, which has the same size as the input image. The idea of this algorithm is to add a feature point to the final feature

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

point list only when its position is not masked out in the mask image. If it is added to the list, all neighboring pixels which are within the specified minimum distance are masked out in the mask image. As the standard algorithm, it starts at the first feature with maximal cornerness and moves towards features with lower cornerness until it reaches the end of the input vector (which we got in step 4) or until enough new feature points have been added (as specified by the maximum allowed number of feature points). It is possible to choose automatically the faster minimum-distance-enforcement algorithm (standard vs. alternative), based on the minimum distance, the number of input features and the maximal number of features.

Feature Point Tracking Implementation

Unlike the feature detection, all the tracking steps have been packed inside only one complex kernel, so that a significant part of the data could be read once and then kept in the shared memory. In each pyramid level, each thread does the calculations for exactly one feature point.

Since bilinear interpolation is essential for achieving sub-pixel accuracy in the tracking algorithm, texture memory has been used to store the image pyramids for the images *I* and *J*. This allows to achieve sub-pixel reads at the cost of a normal read access.

One of the most critical optimizations was to reduce the number of necessary texture fetches, especially in the most inner loop of the algorithm, where the mismatch vector *b* is being calculated, as those are the most time-consuming operations.

Another important issue in the optimization process involves minimizing both shared memory and register usage, while not allowing the compiler to place often used variables in the local memory space, which has a very high access latency just like global memory.

Also, finding the best compromise between the number of registers per thread block, the amount of shared memory used and the number of threads per block is very troublesome and requires a lot of experiments. In most cases a good idea is to set the number of threads per block to 128 or 256 as those configurations allow the full utilization of multiprocessors. For the tracking kernel this number had to be reduced as each thread requires a lot of shared memory and registers on its own. Furthermore one should remember that in many cases the overall number of features to track, and therefore threads, is relatively low, like less than a few thousands, so the number of threads per block should be reduced even more. For example if there are a thousand threads in the tracking kernel, it's not efficient to set it to 256 as there would be only four blocks for four multiprocessors. In that case a GPU like the Geforce 280GTX, which has 30 multiprocessors, would use only 13% of its computing resources, as a single thread block can never be split between different multiprocessors.

Note that in contrast to the GPU KLT implementations presented in [Sin06] and [Zac08], we do not skip specific levels of the image pyramid. Furthermore, we do a convergence test after every iteration instead of employing a fixed number of iterations.

6. EXPERIMENTS AND RESULTS

This section describes experiments done with the CUDA KLT implementation. We compare the CUDA implementation with the corresponding function in the OpenCV library in terms of quality and speed. Note that the OpenCV library internally uses the Intel Performance Primitives (IPP) library and OpenMP for performance reasons. The runtime measurements were done on a 2.4 GHz Intel Xeon Quad-Core machine, equipped with a NVIDIA Geforce GX280 GPU.

Quality Tests

In Figure 3 a comparison of the detected points by both routines is given. One sees that there are neither green nor red points, indicating that the CUDA implementation of feature detection gives the same feature points as the OpenCV routine.

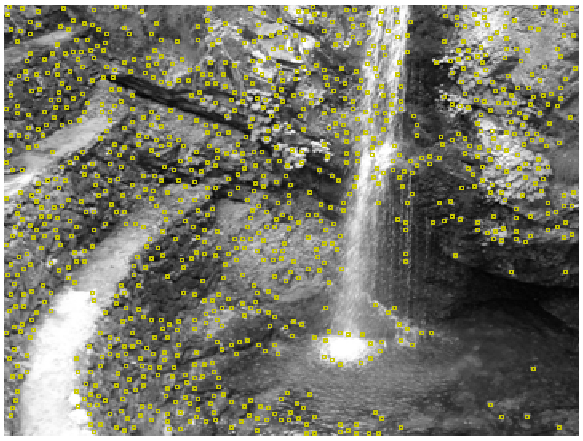


Figure 3: Quality results: Features detected by: Red = OpenCV, Green = CUDA, Yellow = Both.

The results of the tracking algorithm are shown in Figure 4. For each feature point its estimated motion vector is drawn in. Only a small percentage of the feature points has different motion vectors. These differences might arise mainly due to the usage of a more precise 5 x 5 Gaussian convolution kernel for creating the image pyramids in the CUDA implementation. For most correctly tracked feature points both implementations give similar results.

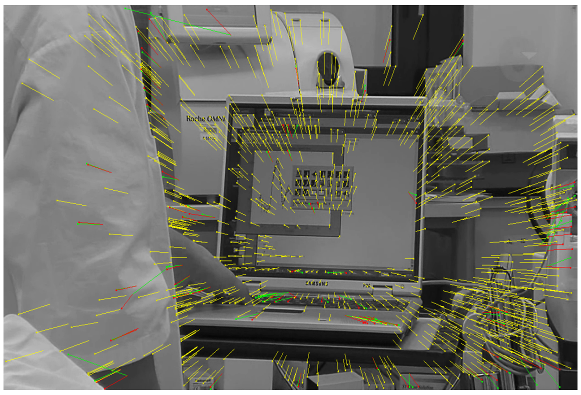


Figure 4: Quality results: Features tracked by: Red = OpenCV, Green = CUDA, Yellow = Both.

Runtime Tests

The various parts of the KLT algorithm depend on different parameters. All the tests were done using the parameters from Table 2, if not specified otherwise. Note that HD 1080p denotes an image resolution of 1920 x 1080 pixels, HD 720p denotes 1280 x 720 pixels and SD 720 x 576 pixels.

Video format:	HD 1080p (1920x1080)
Maximum # features:	10000
Quality level:	5 %
Minimum distance:	6 pixel
Window size (detection):	5 x 5
Enforce min. dist. algorithm:	Automatic
Pyramid levels:	6
Accuracy threshold	0.03 pixel
Maximum iterations:	10
Window size (tracking):	5 x 5

Table 2: Default parameters used for experiments.

Figure 5 shows the runtime for the first four steps of the feature detection algorithm for different image resolutions. Experiments have shown that the runtime is practically independent on any parameters apart from the image resolution.

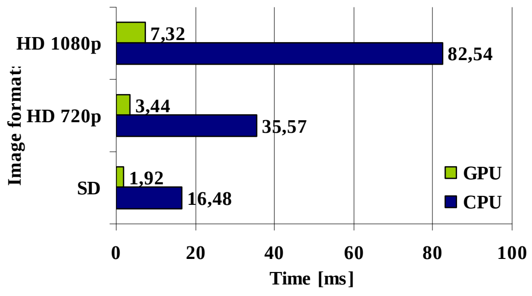


Figure 5: Runtime of the feature point detection (without minimum-distance-enforcement) for different image resolutions.

The runtime for the enforcement of the minimum distance is shown in Figure 6. It depends mainly on the number of features.

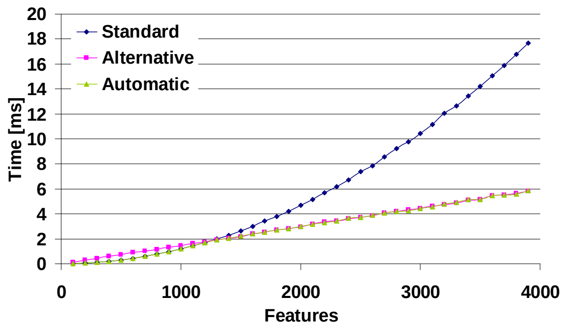


Figure 6: Runtime of the minimum distance enforcement

The runtime measurements in Figures 7 and 8 show the feature point tracking results for SD and HD 1080p material for different numbers of feature points.

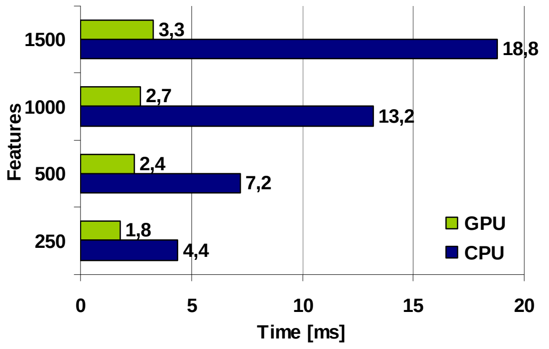


Figure 7: Runtime of the feature point tracking for different number of features for SD (720x576).

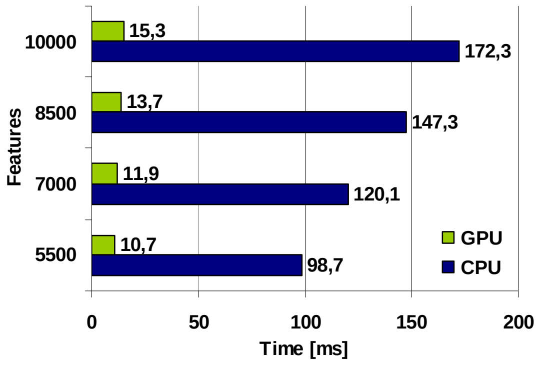


Figure 8: Runtime of the feature point tracking for different number of features for HD 1080p (1920x1080).

Some experiments were done with different window sizes for the feature point tracking and are shown in Figure 9. One can see that increasing the window size results in a significant increase in runtime, so one should use the smallest possible window size. For most cases, a window size of 5 x 5 should suffice for feature point tracking.

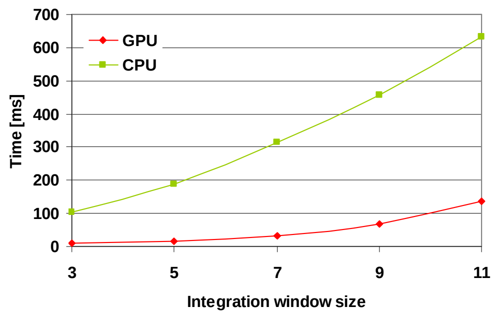


Figure 9: Runtime of the feature point tracking for different window sizes for HD 1080p (1920x1080).

Finally, Figure 10 shows the overall runtime of the KLT algorithm (feature point detection & tracking).

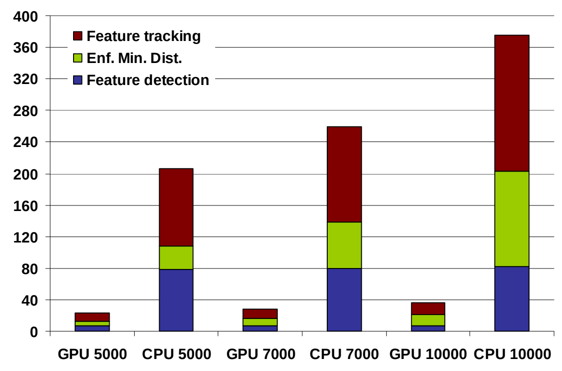


Figure 10: Overall runtime for different numbers of feature points for HD 1080p (1920x1080).

Overall, the CUDA implementation achieves a speedup of approximately 5 – 10. The speedup is higher for larger images and more feature points. This might be due to better utilization of the GPU's processing capabilities.

7. CONCLUSION

The well known KLT algorithm was ported to the GPU using CUDA. Experiments were done which show that the GPU implementation has the same quality as the corresponding CPU (OpenCV) routine, but runs significantly faster (approximately 5 to 10 times). The usage of the GPU makes it possible to track several thousands of feature points on Full-HD material in realtime (>25fps).

8. ACKNOWLEDGMENTS

This work has been funded partially under the 7th Framework program of the European Union within the project "2020 3D Media – Spatial Sound and Vision" (ICT-FP7-215475) and by the Austrian Federal Ministry for Transport, Innovation and Technology within the project "VAN-DAL".

9. REFERENCES

[Che08] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, K. Skadron, A performance study of general-purpose applications on graphics processors using CUDA, Journal of Parallel and Distributed Computing, 2008

[Kan06] N.K. Kanhere, S.T. Birchfield, W. A. Sarasua, Vehicle Segmentation and Tracking in the Presence of Occlusions, Transportation Research Board Annual Meeting, 2006

[Koc99] R. Koch, B. Heigl, M. Pollefeys, L. V. Gool, H. Niemann, Calibration of Hand-held camera sequences for plenoptic modeling, Proceedings of the International Conference on Computer Vision, pp. 585-591, 1999

[Luc81] B.D. Lucas, T. Kanade, An Iterative Image Registration Technique with an Application To Stereo Vision, Joint Conference on Artificial Intelligence, pp. 674-679, 1981

[Lyp07] Y. Lypetskyy, Robust pedestrian detection and tracking in crowded scenes, Proceedings of the SPIE, Volume 6764, 2007

[Ohm08] J. Ohmer, N. Redding, GPU-Accelerated KLT Tracking with Monte-Carlo-Based Feature Reselection, Digital Image Computing: Techniques and Applications, 2008

[Tom91] C. Tomasi, T. Kanade, Detection and Tracking of Point Features, Technical Report CMU-CS-91-132, Carnegie Mellon University, 1991

[Sin08] S. Sinha, J. Frahm, M. Pollefeys, Y. Genc, GPU-Based Video Feature Tracking and Matching, Technical Report 06-012, Department of Computer Science, UNC Chapel Hill, 2006

[Shi94] J. Shi, C. Tomasi, Good Features to Track, IEEE Conference on Computer Vision and Pattern Recognition, pp. 593-600, 1994

[Zac08] C. Zach, D. Gallup, J.M. Frahm, Fast gain-adaptive KLT tracking on the GPU, CVPR Workshop on Visual Computer Vision on GPUs, 2008

Preprocessing of microscopy images via Shannon's entropy on GPU

Jan Urban
Institute of Physical Biology
Zamek 136
373 33, Nove Hrady
Czech Republic
urban@greentech.cz

Jan Vanek
Institute of Physical Biology
Zamek 136
373 33, Nove Hrady
Czech Republic
vanekyj@kky.zcu.cz

Keywords
Image analysis, phase-contrast microscopy, Shannon’s entropy, GPU.

FULL PAPER

1. INTRODUCTION

This document presents GPU speed-up on entropy contribution as image preprocessing method based on Shannon's entropy. The method is developed specially for microscopy images captured in phase-contrast mode. But it can be used in many others applications. Illustrative description of using entropy is proposed in the paper and advantages are discussed. Performance of individual methods is illustrated. Finally, implementation on graphics cards to overpass higher computation requirements of the algorithm is described. The total speed-up of the processing is about 3600x.

Observation of living biological systems (cell culture) in long time as non-invasive technique via microphotography (time laps) produce huge amount of data (images, frames) for subsequent analyzing steps. Monitoring the live cells and their behavior in time consist of cell to background segmentation, segmentation of individual cells, and identification (parameterization) of cell events (division, fusion, death, communication, etc.).

Those processing demands high computation power especially for proper results in reasonably amount of time. For assign the area of interest it is necessary to locate the time changes as changes between captured images using automated image analysis and identify the events in sense of biological terms. There are plenty of different methods for changes evaluation available and described in the literature [1, 2, 3, 4, 5, 6, 7, 8]. We propose a novel idea of entropy fluxes between captured images set, even between non-immediately ensuing. This method is based on information contribution of one pixel [9] to the information content in Shannon's meaning. In fact, it represents how many observable information was transferred in time to/from the pixel location. Because of high computation time of this task, the algorithm should be optimized for graphical processing unit (GPU). The tasks of image analysis are generally and semi-easily prepared for parallelization because of pixel independence on the rest of the image in major task steps.

2. Methods

Shannon's entropy from information theory describes surprise of occurrence of given events v, where each event v may occurs with different probability p_v. The total amount of information (entropy) is the average of the information of the individual events weighted by the probabilities of their occurrences [10, 11, 12]. Definition of Shannon's entropy is

$$S=-\sum\left[p_v \log_2\left(p_v\right)\right] . \tag{1}$$

In the image analysis, we are able to measure the information as Shannon's entropy where instead of unknown probability distribution is used normalised histogram function $H(v)$.

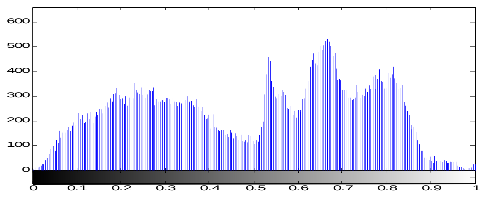


Figure 1: Histogram function before normalisation.

Therefore, probability p_v of event v means amount of pixels in the image (or selected part of the image) with intensity value assigned to the event v. This amount is divided by the total amount of pixel in the image (or selected part of the image) to fulfil the condition

$$\sum p_v = 1 \ . \qquad (2)$$

In the pixel contribution to the information content [9] is computed individual Shannon's entropy for each pixel *f(i,j)* in the image. Cross of the whole row *i* and whole column *j* in the image, where the current pixel position is located, was chosen as part of the image. But, there should be any type and size of selected part for pixel contribution computation, depended somehow on the objects in the image(s). The value of pixel *f(i,j)* is counted only once.

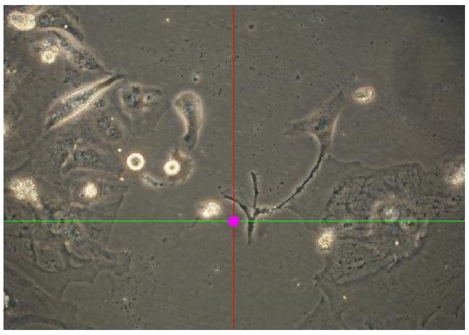


Figure 2: Cross for current pixel f(i,j).

During evaluation of pixel contribution is counted difference between entropy of selected part with current pixel and entropy of the same selected part without current pixel. Resulted value is the measure of current pixel contribution to the entropy of selected part of image. The histogram function for the cross with centre pixel is

$$H(v)_{(i,j)} = H(v)_i + H(v)_j - h(v)_{(i,j)} \ . \qquad (3)$$

And the histogram function without the center pixel is

$$H(v)_{(i,j)} = H(v)_i + H(v)_j - 2\mathrm{h}(v)_{(i,j)} \ , \qquad (4)$$

where $H(v)_i$ and $H(v)_j$ are precomputed histograms of row *i* and column *j*, respectively.

3. RESULTS

One of the powerful tools is based on evaluation of information entropy using the Shannon equation. Although the formula is well known, there are still many ways how to use it.

To prove the usability of developed approach, we performed the tests on several biological samples from different experiments (Figure 1). For all kinds of images the algorithm performance is very good.

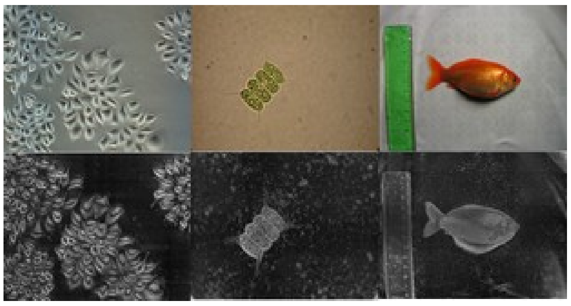


Figure 3. From left to right: *HeLa* cell line experiment using phase contrast; Green alga *Scenedesmus* using bright field microscopy; *Chinese goldfish* digital camera picture

4. IMPLEMENTATION ON GPU

Currently, two high-level GPU programming technologies from both main manufactures are usable. Brook+ from AMD [16] and CUDA from NVIDIA [18]. The algorithm was done in CUDA because of NVIDIA hardware. GPU programming is not so easy especially if maximum speed is necessary. But the final speed of implementation satisfies more sophisticated programming style.

The key to high performance of implementation is to fit the algorithm to GPU highly-parallel architecture. The architecture of NVIDIA GPUs is illustrated on figure 5.

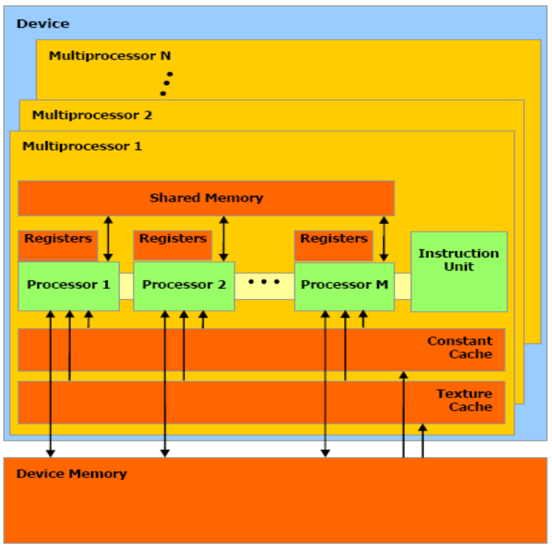


Figure 4: NVIDIA GPU architecture with a set of multiprocessors with on chip shared memory.

For the architecture the double-hierarchy is typical. Whole GPU is the set of multiprocessors as well as the multiprocessor (MP) is a set of eight scalar processors (SP). All multiprocessors can access data in the device memory. For reading they can employ texture or constant cache. SPs inside one multiprocessor can utilise excepting its registers also joint shared cache. Read and write accesses can be synchronised and the SPs inside one multiprocessors can cooperate this way. The survey of all memory kinds are listed for betted understanding:

- **Host (CPU) memory** - "normal" memory where all data have to be prepared before transfer to GPU memory through PCI-Express bus. The results are transferred back from GPU to host memory after computation. It is faster to transfer less amount of larger memory blocks than transfer high number of small blocks. Keep in mind that the bus is relatively very slow and could be a bottle-neck of the computation performance.
- **Device (GPU, global) memory** - main memory installed on graphics card. It has high delay therefore the implementation has to look at this fact. Random read/write of single data-types affects the performance a lot. Using block read/write ("coalesced access") is necessary. The second option is to use constant or texture cache.
- **Constant cache** - limited amount of this kind of memory can be used for reading. It is advisable to use it for example for look-in tables.
- **Texture cache** - read-only data in device memory can be cached via texture cache. It can be allocated either linear memory or in 2D manner. It is advisable to use it for all data which are read-only.
- **Shared memory** - limited amount of on-chip memory which is shader-clocked and it belongs to individual multiprocessor. It is accessible only for SPs of this multiprocessor. The implementation should avoid "bank conflicts" which reduce performance. Appropriate using of this kind of memory can be a key part of high-performance implementation.
- **Registry** - memory which belongs to individual SPs. They are used to store the internal variables.

CUDA data-parallel programming model is based on the hardware double-hierarchy. The data have to be split into algorithmically independent parts on two levels. At the first level the data are split into grid of blocks. Each block is processed with the same algorithm which is called "kernel". During processing of the block several number of thread is running. It is the second level of the hierarchy (it is illustrated in fig. 6). All threads which evaluate one block are running on one multiprocessor and they can utilise its shared memory for data interchange.

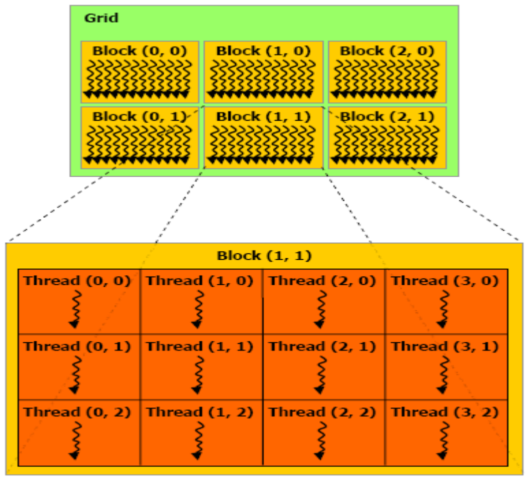


Figure 5: CUDA double-hierarchy data-parallel programming model.

To manage this programming model, SIMT (single-instruction multiple thread) architecture is employed. The multiprocessor maps each thread to one SP core, and each scalar thread executes independently with its own instruction address and register state. The multiprocessors threads are executed in parallel groups. The programmer has to implement its algorithm this parallel way too. At least 32 threads should do the same work.

In our task - entropy flux calculation - the data parallelisation is strait-forward as well as in other image processing tasks where pixels can be evaluated independently. If the histograms for all rows and all columns are precomputed, the algorithm for single pixel is following:

- Load and add the histograms of row *i* and column *j*.
- Normalise histogram.
- Compute entropy value with current pixel.
- Load pixel values from the current as well as the next-image pixel.
- Subtract the current pixel values and add the next-image pixel values from the histogram.
- Compute entropy value with next-image pixel.
- Store the resulted entropy difference into device memory.

Before entropy computation itself, the histograms are pre-computed. Each pixel of the image represents one block of the grid. In each grid 128 threads are executed. The histogram vectors are 256 long, therefore the entropy intra-sum-values are computed in two steps. The sum itself is calculated in semi-parallel manner [GPU-sum] in seven steps. The optimization of implementation in CUDA is not trivial but the performance could be highly affected by non-optimal implementation.

Times of processing 2Mpix and 6Mpix images and cumulative speed-ups for all implementations are shown in Table 1. All tests were done on computer with Intel Core2Duo 2.4GHz CPU and NVIDIA GeForce 9800GTX+ GPU. CPU times assume single-core versions. The total speed-up of the processing is about 3600x with this configuration. So what used to take hours before, now cause come in matter of seconds.

Version	**Elapsed Time**		**Cumulative speed-up**
	2Mpix	**6Mpix**	
Matlab	27 min	> 2 h	-
Matlab optim.	136 s	13 min	12x
C++	21.8 s	128 s	72x
Framewave	11.2 s	64 s	150x
CUDA GPU	0.45 s	2.75s	3600x

Table 1: Processing times and speed-ups

5. CONCLUSION

Useful extension of pixel contribution based on Shannon's entropy was described and its features were discussed. Our approach of entropy fluxes allows to identify the location in the image set with relevant changes of information content. Used equation is very simple, but computational time for each pixel is time consuming. Therefore, we propose to using parallelisation on graphics cards. Finally, optimization of the algorithm was described and enormous speed-up was achieved. Implementation on GPU was done in CUDA. However, it could be also simply rewritten in OpenCL. In possibility of future release of Intel Larrabee platform, the algorithm could efficiently use its novel cache architecture for sharing of precomputed histograms.

The algorithm should be used for other images with objects on more-less simple background.

6. ACKNOWLEDGMENTS

This work was supported by grant HCTFOOD A/CZ0046/1/0008 of EEA funds and GAJU grant 091/2008/P of Grant agency of South Bohemia.

7. REFERENCES

[1] Sonka M., Hlavac V., Boyle R.: „Image Processing, Analysis and Machine Vision“, *Brooks/Cole Publishing Company*, 1999

[2] Gonzales R. C., Woods R. E.: „Digital Image Processing“, *Addison-Wesley Publishing.,* 1992

[3] Yang J. Lu W. Waibel A.: :„Skin-Color Modeling and Adaptation“, *Proceedings of the Third Asian Conference on Computer Vision-Volume II,* 1998

[4] Otsu N.: „A Threshold Selection Method from Gray-Level Histogram," *IEEE Trans. On Systems, Man, and Cybernetics SMC-9, pp. 62--66*, 1979

[6] Beucher S., Lantuejoui C.: „Use of Watershed in contour detection“, *International Workshop on image processing, real-time edge and motion detection/estimation, Rennes, France*, 1979.

[7] Vanek J., Urban J., Gardian Z.: „Automated detection of photosystems II in electron microscope photographs“, *Technical Computing Prague,* 2006.

[8] Carpenter et al.: „CellProfiler: image analysis software for identifying and quantifying cell phenotypes “, *Genome Biology, 7:R100*, 2006.

[9] Urban J., Vanek J., Štys D.: „Preprocessing of microscopy images via Shannon's entropy”, In proc. Of *PRIP, Minsk, Belarus,* 2009

[10] C. E. Shannon, ``A mathematical theory of communication," *Bell System Technical Journal,* vol. 27, pp. 379-423 and 623-656, 1948.

[11] Hatley J.V.: *Bell System Technical Journal 7,* 535, 1928

[12] P.Jizba and T.Arimitsu: The world according to Renyi: Thermodynamics of multifractal systems, *Analytical Physics. 312, 17 — 59,* 2004

[13] Vincent L., „Morphological grayscale reconstruction in image analysis: applications and efficient alghorithms“, *IEEE Transactions on Image Processing, 176-201,* 1993

[14] MATLAB software, *www.mathworks.com*, The Mathworks, Natick, Massachusetts, USA.

[15] Framewave, AMD Performance Library *developer.amd.com/cpu/libraries/framewave*

[16] General-Purpose Computation Using GPUs, www.gpgpu.org

[17] AMD/ATI Stream computing SDK, *ati.amd.com/technology/streamcomputing*

[18] NVIDIA CUDA gpgpu development tools, www.nvidia.com/cuda

THE BPT ALGORITHM (BRIANCHON – POINT – TRIANGLE) - DETECTING CONICAL CURVES IN RASTER GRAPHICS

Krzysztof T. TYTKOWSKI
The Silesian University of Technology
Geometry and Engineering Graphics Centre – RJM-4
POLAND, 44-100 Gliwice, ul. Krzywoustego 7
E-mail krzysztof.tytkowski.@polsl.pl

Keywords
algorithm, Brianchon's theorem, conical curves, detection

1. INTRODUCTION

Many aspects of technology require determination of curves e.g. 3D restitution of objects etc. In the past pictures obtained in photochemical way were used and then basing on them characteristic points were determined. The precision/ depended on the zoom which was used as well the material used for picture. These pictures were made on material of low speed and thanks to that the pictures were in focus which allowed precise determination of characteristic points. Introduction of digital photos facilitated the process significantly and basing on that new applications occurred e.g. vision systems. Accuracy/ precision in case of digital picture depends on its size (the size of matrix). With analog picture zoomed to format e.g. 0.6x0.4 m the precision/ could reach up to 0.001 m. Having similar precision/ between digital picture and analog picture requires 6000x4000 pixels resolution. Kodak KAF-50100 8176 x 6132 pixels, (which gives 50 million pixels) [Kod01] is the ready made matrix of biggest resolution which is currently produced.

Processing data is the next aspect. Recognition in analytic and descriptive way is possible. These two methods are probable in analog picture as well but then it would be necessary to determine e.g. coordinates of characteristic points. It can be done by scanning or reading coordinates from analog picture. Using descriptive methods it is not compulsory to determine points coordinates but to use e.g. tangent to curve and curve properties. This different approach results in completely different solutions to the problem of curve recognition e.g. of conical curve.

As regards analytic methods it is necessary to obtain coordinates of points belonging to curve. In case of conical curve the most general solution is in the form:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \qquad (1)$$

Assuming that $F \neq 0$, a set of quadratic equations for points: $Q_1(x_1,y_1)$, $Q_2(x_2,y_2)$, $Q_3(x_3,y_3)$, $Q_4(x_4,y_4)$, $Q_5(x_5,y_5)$ is necessary in order to solve it

$$\begin{aligned} Ax_1^2 + Bx_1y_1 + Cy_1^2 + Dx_1 + Ey_1 &= 1 \\ Ax_2^2 + Bx_2y_2 + Cy_2^2 + Dx_2 + Ey_2 &= 1 \\ Ax_3^2 + Bx_3y_3 + Cy_3^2 + Dx_3 + Ey_3 &= 1 \qquad (2) \\ Ax_4^2 + Bx_4y_4 + Cy_4^2 + Dx_4 + Ey_4 &= 1 \\ Ax_5^2 + Bx_5y_5 + Cy_5^2 + Dx_5 + Ey_5 &= 1 \end{aligned}$$

The solution of that set of equations is obvious but complicated, and requires many mathematical operations.

Analytic methods as well as some descriptive methods use points which e.g. belong to curve. In methods based on descriptive constructions it is also possible to use tangents to the conic. The method of obtaining information on curve from the found tangent to the curve is easier and more exact than finding point on a curve. In case of a point it is the error of determination of each coordinate and in horizontal or vertical tangent the error is connected with one coordinate.

1.1 History

The research on conical curves started in ancient Greece. The first definition of conics was coined by Menaechmus (Μέναιχμος, 380–320 BC), who was the disciple of Plato and Eudoxus.

He used parabola when solving a problem of doubling cube volume (which is the solution of equation $x^3=2$). He introduced a notion 'section of a right-angled cone' for ellipse 'section of an acute-angled cone', and for hyperbola 'section of an obtuse-angled cone' [Loc61]. Euclid (Εὐκλείδης fl. 300 BC) wrote four books on conics. Archimedes of Syracuse (Ἀρχιμήδης; c. 287 BC – c. 212 BC) using the method of exhaustion worked on parabola. When solving the problem of doubling cube volume (which is the solution of equation he used parabolas. He used the following notion for parabola'section of a right-angled. Cone', for ellipse 'section of an acute-angled cone', and for hyperbola 'section of an obtuse-angled cone' [Loc61]. Euclid (Εὐκλείδης fl. 300 BC) also wrote four books on conics. Archimedes of Syracuse (Ἀρχιμήδης; c. 287 BC – c. 212 BC), using the method of exhaustion, dealt with calculation of area under parabola and ellipse arc. Thanks to Apollonius of Perga (Ἀπολλώνιος, ca. 262 BC–ca. 190 BC) we have the name parabola (παραβολή ' 'equality', 'an exact comparison'.), ellipse (ἔλλειψις, a 'falling short'), and hyperbola (ὑπερβολή, 'over-thrown' or 'excessive'). He wrote eight volume Conic Sections and researched the properties of conics. Pappus of Alexandria (Πάππος ὁ Ἀλεξανδρεύς, c. 290 – c. 350), introduced the notion of focus of a conic and directrix [Ivo41].

Translation of Greek works into Arabic resulted in preservation of these achievements as for example Omar Khayyám (عمر خیام; 1048 AD in Neyshapur – 1123 AD in Neyshapur,), used conics for solving algebraic equations.

Johann Kepler (27th December 1571 in Weil der Stadt, Württemberg – 15th November 1630 in Regensburg) developed the theory of conics adding the 'principle of continuity'. Desargues (21st February or 2nd March 1591 in Lyon – October 1661 in Lyon) and Blaise Pascal (19th June 1623 in Clermont – 19th August 1662 in Paris) formed the foundation of projective geometry. The following scientists are connected with ellipse: Johannes Kepler and Tyge Ottesen Brahe (14th December 1546 in Knutstorp, Skane – 24th October 1601 in Prague, Bohemia), - orbit of Mars was elliptical, Kepler was the first one to use the notion of 'focus'. René Descartes (31st March 1596 in La Haye ,Touraine – 11th February 1650 in Stockholm) carried research on conics by means of algebraic methods. These are only fractions of facts and names connected with research and application of conics.

2. USING TANGENTS

Determination of tangents to curve in case of raster picture in chosen directions (horizontal and vertical) is much easier and more exact than finding points on that curve. In case of raster picture some directions are special i.e. horizontal and vertical line.

The next four directions where it is easy to determine tangents are tangent at the angle of ¼π to horizontal and vertical. Having six tangents to the curie at our disposal it is possible to determine Brianchon[1]'s point P_B (Fig. 1) [Cox69], [Veb10], [Lin22], [Mat14], [Way17]. In order to eliminate the mistake of accepting curve as conic, for the analysis it is necessary to choose six out of eight tangents. It gives 28 cases, which significantly rises credibility of the method. If lines do not intersect in one point we deal with a triangle. The size of the triangle shows how given curve diverges from a tangent.

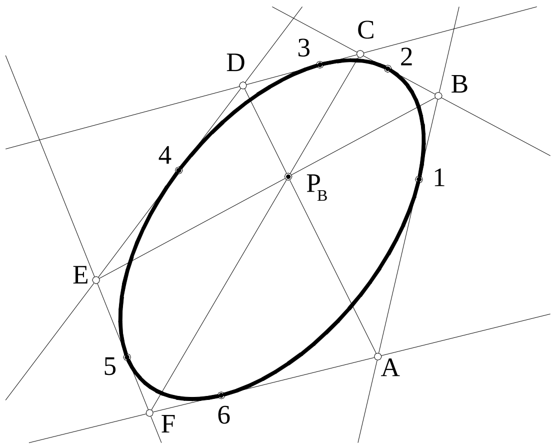


Fig. 1 Determination of Brianchon's point

3. PRINCIPLE OF DETECTING TANGENTS

For determination of eight tangents the properties of raster picture have been used. Tangents in both horizontal and vertical direction can be easily determined. In this way four tangents are obtained. The remaining tangents are a little more difficult to determine. Tangents slanted at the angler of ¼π to the previous ones have been used in the algorithm (Fig. 2). Knowing directions of tangents (previously

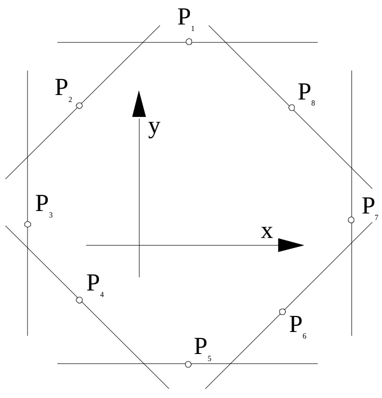


Fig. 2 Determination of points

[1] Charles Julien Brianchon (19th December – 1783 in Sèvres, 29th April 1864 in Versailles, France), published his theorem in 1806 in the Journal de 1'Ecole Polytechnique, Vol. XIII, [Emc05] and afterwards repeated in his *Mémoire sur les lignes du second ordre* (Paris, 1817) [Cre85].

assumed) it is enough to find one point belonging to a given tangent. As it can be easily seen not all coordinates must be determined because points are on proper straight lines and thus for points P_1 and P_5 coordinate y is the only one important, whereas for points P_3 and P_7 it is only coordinate x which is essential. For points P_2,P_4,P_6,P_8 both coordinates are significant but we know the angle of inclination to the straight line.

In order to find point P_1, and in fact coordinate y_1, it is necessary to find point which has the biggest value of coordinate y. The same refers to point P_5 but it will be the smallest value of y of curve point.

Next point P_3, and to be precise coordinate x_3, will be easy to find since it is curie point of minimal value of coordinate x, and maximum value of coordinate x will be for point P_7. Point P_2 is a point on a line which is slant to axis x at the angle of ¼π. The points on that line are characterized by the fact that difference of coordinates x and y is of minimal value. Point P_6 on tangent, as the remaining points on the line, has maximal value of that difference. Similarly to point P_4 but for all points on tangent the sum of coordinates x and y reaches minimum and for P_8 the sum reaches maximum.

4. THE ALGORITHM OF PIXELS DETECTION

Assuming that the analyzed picture is of size v pixels in vertical and h pixels in horizontal and information on e.g. color is in the box of o[v,h] then this algorithm could be as follows:

```
var:
xmin=v, ymin=h, xmax=1, ymax=1;
sumin=v+h, dimin=v-h, sumax=2, dimax=0;
begin
  for I=h do
    for k=v do
      if o(i,k)=color then
        su=x+y;
        di=x-y;
        if i<xmin then xmin=i;
        if k<ymin then ymin=k;
        if i>xmax then xmax=i;
        if k>ymax then ymax=k;
        if su<sumin then sumin=su;
        if di<dimin then dimin=di;
        if su>sumax then summax=su;
        if di>dimax then dimax=di;
    k=k+1;
  i=i+1;
end.
```

5. FINDING POINTS OF TANGENTS INTERSECTION

Since raster picture forms an approximation of curve and in fact we do not know where exactly curve points are, but we only know area in form of pixels where they occur; therefore, finding points of tangents' intersection is rough. Depending on which tangents we choose point of intersection it can be of type a, b, or c (Fig. 3) as well as it can also be incorrect point. The areas where such point can occur are rhomboids (sometimes squares). If the solution was looked for with a set of equations then each coordinate should be considered with some accuracy, depending on the size of pixels.

It is possible to use two ways: circle estimation or point coordinates resulting from rhomboid (Fig 3).

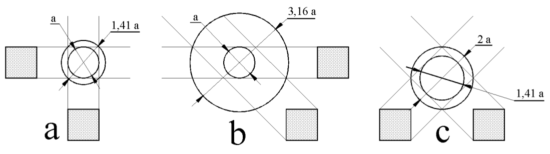


Fig. 3 Types of areas depending on types of straight line

The first method is based on inscribed or described circles, the latter one which is more exact is based on extension of rhomboid sides. The first method gives apparent area, which is especially visible in type b where there is neither described nor inscribed circle since the circle intersects only two out of four vertexes or is tangent only to two out of four sides.

Therefore lines analysis which are extension of rhomboid sides, is a more accurate method. Tangents intersection passing pixels pairs P_1P_3, P_3P_5, P_5P_7, P_7P_1 will be of type a , pairs P_2P_4, P_4P_6, P_6P_8, P_8P_3. type c, pairs P_1P_5, P_2P_5, P_3P_7, P_4P_8 are incorrect points, and the remaining cases are of type b.

Determination if such location of lines, incorporated between lines resulting from extension of pixels outlines (tangents) so that they intersect in Brianchon's point, is possible requires checking if there is common area for outlines of three lines (Fig. 4– such an area where one can find point P_B is GFK triangle).

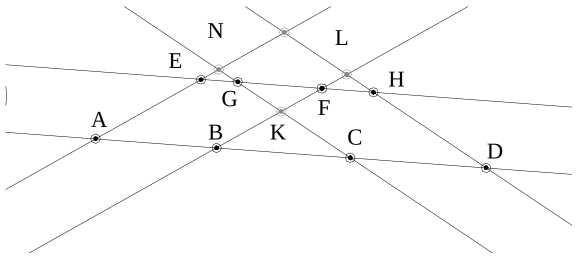


Fig. 4 GFK Brianchon's triangle

In order to check if such area is probable it is necessary to consider all possible cases. Conditioned instructions can be used for that purpose but they have to be multilayer. Another way to determine if point P_B exists is using anharmonic ratio.

In order to state if there is a possibility of finding Brianchon's point it is necessary to have common area for the outline of three liners. This area in a form of a triangle is called Brianchon's triangle which is an equivalent of Brianchon's point in raster graphics. Two anharmonic ratios e.g. created in intersection of lines forming one outline with lines determining the remaining two outlines, resulting from 'points' where tangents intersect, will be used to state if such a triangle exists. Univocal determination if it is possible that tangents will intersect in one point needs researching two different anharmonic ratios [Cox69], [Veb10], [Lin22], [Mat14], [Way17]. Let the first one be four ABCD and let us calculate value of k_1

$$k_1 = \frac{AC}{BC} / \frac{AD}{BD} \qquad (3)$$

Where : AC is the distance of point C to point A
BC is the distance of point C to point B
AD is the distance of point D to point A
BD is the distance of point D to point B

5.1 Possible location of points ABCD anharmonic ratio

Assuring that point A, which is on the line, has smaller coordinate x from point B, and point C from point D; the possible location of these points on the line is as follows: ABCD, ACBD, ACDB, CADB, CDAB, CABD (Fig. 5)

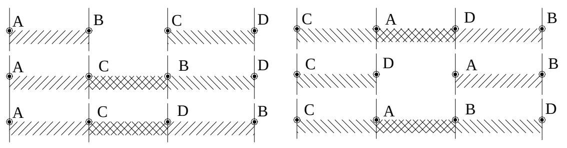


Fig. 5 Possible mutual location of points ABCD

It should be noticed that if points ABCD are different then common area will not occur for two sets ABCD and CDAB. In other cases common area will exist. In order to determine quickly which case are we dealing with we will use anharmonic ratio.

On the line we assume a point from which distances to particular points will be measured. The distance between this assumed point and point A is determined as a. For the remaining points their distance from base point is given in relation to point A, as thus the distance to point B is b=a+l.

5.1.1 abcd – lack of common area

$$\frac{AC}{BC} \Big/ \frac{AD}{BD} = k_1 \qquad (4)$$

With this mutual location of points, the distances reach respectively:

$$b = a + l_1, \quad c = b + l_2, \quad d = c + l_3 \qquad (5)$$

$$k_1 = \left(\frac{c-a}{c-b}\right)\left(\frac{d-b}{d-a}\right) \qquad (6)$$

Putting dependencies (5) to the equation (6) we get:

$$k_1 = \frac{(a + l_1 + l_2 - a)(a + l_1 + l_2 + l_3 - a)}{(a + l_1 + l_2 - a - l_1)(a + l_1 + l_2 + l_3 - a - l_1)} \qquad (7)$$

After necessary simplifications we obtain

$$k_1 = \frac{(l_1 + l_2)(l_1 + l_2 + l_3)}{l_2(l_2 + l_3)} > 1 \qquad (8)$$

Therefore, for points location in the ABCD order $k_1>1$

5.1.2 cdab – lack of common area

In this situation distances equal respectively:

$$b = a + l_1, \quad c = d - l_3, \quad d = a - l_2 \qquad (9)$$

Putting dependencies (9) to the equation (6) we get:

$$k_1 = \frac{(a - l_2 - l_3 - a)(a - l_2 - a - l_1)}{(a - l_2 - l_3 - a - l_1)(a - l_2 - a)} \qquad (10)$$

After simplifications

$$k_1 = \frac{l_1 l_2 + l_1 l_3 + l_2 l_2 + l_2 l_3}{l_1 l_2 + l_2 l_2 + l_2 l_3} > 1 \qquad (11)$$

Similarly to the previous case for points location in the CDAB order $k_1>1$

5.1.3 acbd – areas have common part

The points' distance can be recorded as:

$$c = a + l_1, \quad b = c + l_2, \quad d = b + l_3 \qquad (12)$$

Putting dependencies (12) to the equation (6) $_{k1}$ will equal:

$$\frac{(a + l_1 - a)(a + l_1 + l_2 + l_3 - a)}{(a + l_1 - a - l_1 - l_2)(a + l_1 + l_2 + l_3 - a - l_1 - l_2)} \qquad (13)$$

After simplifications we obtain:

$$k_1 = \frac{l_1(l_1 + l_2 + l_3)}{-l_2 l_3} < 0 \qquad (14)$$

If points are in the order ACBD then $k_1 <0$.

5.1.4 CADB – areas have common part

There are the following dependencies between distances

$$c = a - l_1, \quad b = d + l_3, \quad d = a + l_2 \qquad (15)$$

Based on equation (6) and dependencies (15) we get:

$$k_1 = \frac{(a - l_1 - a)(a + l_2 - a - l_2 - l_3)}{(a - l_1 - a - l_2 - l_3)(a + l_2 - a)} \qquad (16)$$

After reduction we acqiure:

$$k_1 = \frac{(-l_1)(-l_3)}{-(l_1 + l_2 + l_3)\, l_2} < 0 \qquad (17)$$

If points are in the order CADB then $k_1 <0$.

5.1.5 acdb – CD is incorporated in AB

The distances between points are equal:

$$c = a + l_1, \quad b = d + l_3, \quad d = c + l_2 \tag{18}$$

Putting (18) to equation (6) gives:

$$k_1 = \frac{(a + l_1 - a)(a + l_1 + l_2 - a - l_1 - l_2 - l_3)}{(a + l_1 - a - l_1 - l_2 - l_3)(a + l_1 + l_2 - a)} \tag{19}$$

After transformations we obtain dependencies for k_1:

$$0 < \frac{-l_1 l_3}{-(l_2 + l_3)(l_1 + l_2)} < 1 \tag{20}$$

If a pair of points CD is inside segment AB then $0<k_1<1$.

5.1.6 cabd – AB is incorporated in CB

In this case distances are:

$$c = a - l_1, \quad b = a + l_2, \quad d = b + l_3 \tag{21}$$

Using equation (6) and dependencies (21) we obtain:

$$k_1 = \frac{(a - l_1 - a)(a + l_2 + l_3 - a - l_2)}{(a - l_1 - a - l_2)(a + l_2 + l_3 - a)} \tag{22}$$

Dependencies for k_1 assume the following form:

$$0 < \frac{-l_1 l_3}{-(l_1 + l_2)(l_2 + l_3)} < 1 \tag{23}$$

If a pair of points AB is inside segment CD then $0<k_1<1$.

Analyzing equations (8),(11),(14),(17),(20) and (23) it should be noticed that in case when consequent occurrence of points excludes a possibility of the existence of an area where Brianchon's point would be of value $k_1>1$.

However, one anharmonic ratio does not provide full answer. If $k_1<1$ then we are sure that Brianchon's triangle is possible. With value $k_1>1$ other anharmonic ratio should be checked e.g.. EFGH (Fig. 4).

$$k_2 = \frac{EF}{GF} / \frac{EH}{GH} \tag{24}$$

Then, analyzing value k_2 we get the answer if Brianchon's triangle exists. If so the analyzed curve is conical curve with the precision of one pixel.

6. BRIANCHON'S TRIANGLES

Choosing in sequence six tangents out of determined eight we check if there is Brianchon's triangle. There are 28 possibilities and with the existence of 28 triangles we can state that all eight tangents are tangent to the conical curve. In case when only seven triangles exist we can state that these seven tangents are tangents to conical curve. If there is only one triangle then only tangents which trace it are conical curves. Naturally, when there is a lack of triangle in all possible sets of eight tangents we can state that none of six out of eight tangents do not fulfill the condition of being tangent to conic line. In remaining cases the value which confirms if we deal with conic line is coefficient Δ_B

$$\Delta_B = \frac{i}{k} \tag{25}$$

where: i – the number of Brianchon's triangles
k – the number of used tangents

7. CONCLUSIONS

BPT algorithm has the following advantages:

- The fact that it is based on tangents their determination in case of raster picture and in chosen directions is fast and easy (it is searching of minimum and maximum of coordinates and their sum and difference). There is no need to solve a set of quadratic equations .
- It is possibile to state in a simple way if a presented curie is a conic or not

However, the following facto are the disadvantages:

- the fact that the algorithm can not cope with a part of a curve since for correct action six tangents are necessary which are at the angle not less than ¼π, so a given section of a curve must be of obtuse value minimum 1½π,
- for constructing e.g. a missing part of a curve other algorithms are necessary but in case of solving a set of equations (2) it is not

Naturally, if we add other tangents apart from the suggested eight it would be also possible to check if a curve section is a part of the conical curve.

8. FUTURE WORK

To perform such a function there is a computer application using a BTP algorithm. The applied program exists already. A predicted testing set will consist of ellipses that variously sized. Images with dimension from 100 to 1500 pixels will contain rotated ellipses (rotation 10 degrees in range 0-90 degrees because of symmetry). Tested will be raster files with antyaliasing and aliasing.

9. REFERENCES

[Cox69] Coxeter H.S.M. Introduction to Geometry, Wiley, 1969

[Veb10] Veblen O., Young J.W. Projective Geometry, Vol 1, Blaisell Publ. Co. 1910

[Lin22] Ling G.H., Wentworth G., Smith D.E.: Elements of Projective Geometry, Ginn and Co 1922

[Mat14] Mathews G.B. Projective Geometry, Longmans, 1914

[Way17] Wayland Bowling L. Projective Geometry, Hill 1917

[Loc61] Lockwood E. H.: A Book of Curve, Cambridge at The University Press, 1961

[Ivo41] Greek Mathematical Works, Ii, Aristarchus To Pappus by Thales (Translator), Ivor Thomas 1941

[Emc05] Emch A.: An introduction to projective geometry and its applications; an analytic and synthetic treatment, New York John Wiley & Sons, London : Chapman & Hall, Limited, 1905

[Cre85] Cremona L. Elements of Projective Geometry, Translated by Leudesdorf C. The Clarendon Press, Oxford, 1885

[Kod09] http://www.kodak.com/global/en/business/ISS/Products/Fullframe/

From Exact Correspondence Data to Conformal Transformations in Closed Form Using Vahlen Matrices

Carsten Cibura
Universiteit van Amsterdam
C.Cibura@uva.nl

Leo Dorst
Universiteit van Amsterdam
L.Dorst@uva.nl

ABSTRACT

We derive a method to determine a conformal transformation in nD in closed form given exact correspondences between data. We show that a minimal dataset needed for correspondence is a localized vector frame and an additional point.
In order to determine the conformal transformation we use the representation of the conformal model of geometric algebra by Vahlen matrices, which helps reduce the problem to purely Euclidean geometric algebra computations, as well as structure the solution into geometrically interpretable components.
We give a closed form solution for the general case of conformal (resp. anti-conformal) transformations, which preserve (resp. reverse) angles locally, as well as for the important special case when it is known that the conformal transformation is a rigid body motion, which preserves angles globally. Rigid body motions are also known as Euclidean transformations.

Keywords: geometric algebra, versors, closed-form solution, conformal mappings, conformal transformations, Euclidean transformations, Vahlen matrices

1 INTRODUCTION

Conformal transformations are transformations that preserve angles locally. They are closely linked to (complex) differentiable mappings and useful in a number of applications in physics and engineering. A notable special case of conformal transformations are rigid body motions or Euclidean transformations. Classically, conformal mappings of the complex plane are studied, but there is a framework that generalizes conformal mappings to spaces of arbitrary dimensions: the conformal model of geometric algebra.

Introduced many decades ago, geometric algebra has gained increasing attention in the recent past by mathematicians, physicists and computer scientists. It facilitates the unified representation of many mathematical and physical problems and promotes the intuitive, because geometrically supported, design and understanding of well-known and new formulas. Its development from a promising framework towards a full calculus [5] has revealed its computational power.

However, well established techniques for solving even basic equations or algorithmic recipes as they might be readily available in, say, classical linear algebra, still seem to be in short supply. Matters are complicated by the non-commutative nature of the fundamental geometric product occurring in the sandwiching versor product which is used to perform transformations on multivectors.

In this paper we use the structuring power of Vahlen matrices to derive a closed form solution for an even conformal versor – the geometric algebra representation of a general conformal transformation – given a minimal set of exact data. To our knowledge such a closed form is new.

Since the given correspondences have to be exact, the solution is suitable for those applications in robotics, computer graphics, differential geometry etc., where a pair of start and target configurations is known precisely and the conformal transformation between the two has to be found.

[7] introduces a numerical method for estimating versors even from noisy data or if more than minimal data is given. Even though a suggestion for efficient implementation is made, this method resorts to classical linear algebra and singular value decomposition (SVD), which introduces extra cost when minimal data is known.

The structure of this paper is as follows. Assuming that the reader is familiar on a basic level with geometric algebra and its conformal model, in section 2 we give a brief introduction and establish the notation we use in this paper. We basically adopt concepts and notations from [4] and [3]. In section 3 we consider a minimal set of data needed to determine an even conformal versor and derive a closed form solution for it. In section 4 we generalize the solution to include odd conformal versors, extend it to arbitrary dimensions and reflect on its general applicability and its limitations. Also we analyze the special case of rigid body motions. In section 5 we present a way to extract a minimal data set from exact point correspondences alone. Section 6 contains concluding remarks and an outlook to future work.

2 PRELIMINARIES

The geometric algebra over a Minkowski space $\mathbb{R}^{n+1,1}$, denoted $\mathscr{G}(\mathbb{R}^{n+1,1})$ is called *conformal geometric algebra* (CGA) when it is used to represent geometric objects and transformations in n-dimensional Euclidean space $\mathbb{R}^{n,0} = \mathbb{R}^n$. In the following we will talk about the elements of the Euclidean vector space $\mathbf{v} \in \mathbb{R}^n$ as *Euclidean vectors* and denote them in bold face. Elements of the Minkowski space $\mathbb{R}^{n+1,1}$ we will refer to as *conformal vectors*. While Euclidean space is a subspace of the Minkowski space and conformal vectors can (and in general do) have Euclidean vector components, we choose this notation to emphasize the geometric significance of purely Euclidean vectors.

In general, the geometric product of any number of vectors results in a *multivector*, the sum of components of different *grade*. The projection of a multivector X onto a specific grade k will be denoted $\langle X \rangle_k$, e.g. $\langle X \rangle_0$ is the scalar part of X, $\langle X \rangle_1$ its vector part etc. If a multivector can be written as the outer product of a number of k vectors, it is called a *k-blade*.

We will denote the manifold of pure k-blades from a geometric algebra over any vector space V by $\mathscr{G}^k(V)$ and introduce the involutions *grade involution* $\widehat{\cdot}$ and *Clifford conjugation* $\bar{\cdot}$ on elements of $\mathscr{G}(V)$ by

$$\widehat{v} = (-1)^k v, \quad v \in \mathscr{G}^k(V), \tag{1}$$

$$\bar{v} = (-1)^{\frac{1}{2}k(k+1)} v, \quad v \in \mathscr{G}^k(V). \tag{2}$$

Note that the grade involution is an automorphism of $\mathscr{G}(V)$ while the Clifford conjugation is an anti-automorphism of $\mathscr{G}(V)$, i.e. $\widehat{vw} = \widehat{v}\widehat{w}$ and $\overline{vw} = \bar{w}\bar{v}$ for $v, w \in \mathscr{G}(V)$.

Finally, we denote the *Clifford group* on a vector space V by

$$\Gamma(V) = \{ s \in \mathscr{G}(V) \,|\, s^{-1} \text{ exists and } sV\widehat{s}^{-1} \in V \}. \tag{3}$$

In order to work with CGA one can introduce a vector basis. A basis is never unique, however, and the most commonly used are the *orthonormal basis* and the *null basis*. The orthonormal basis is given by $\{e_+, e_1, \dots, e_n, e_-\}$ with $e_i^2 = 1$, $i \in \{1 \dots n\} \cup \{+\}$ and $e_-^2 = -1$ as well as $e_i \cdot e_j = 0$ for $i \neq j$. The null basis is given by $\{n_o, \mathbf{e}_1, \dots \mathbf{e}_n, n_\infty\}$ with orthonormal Euclidean vectors $\mathbf{e}_i$, $i = 1, \dots, n$ and $\mathbf{e}_i \cdot n_j = 0$ for $i = 1, \dots, n$, $j = o, \infty$ and n_o and n_∞ defined in terms of the orthonormal basis as

$$n_o = \frac{1}{2}(e_- + e_+), \tag{4}$$

$$n_\infty = e_- - e_+, \tag{5}$$

which implies that $n_o^2 = 0 = n_\infty^2$ and $n_o \cdot n_\infty = -1$. In the following we will use only the null basis.

Conformal vectors can be seen as representing (hyper-)spheres [4] of different (and possibly negative) radii in n-dimensional Euclidean space. Using the inner and outer product we can create a variety of geometric objects [3] like, for example, planes, circles and lines from them, as well as determine incidence and intersection relationships between those. For instance a *conformal point* at Euclidean position $\mathbf{p}$ is represented by the isotropic conformal vector $p = \alpha(n_o + \mathbf{p} + \frac{1}{2}\mathbf{p}^2 n_\infty)$, up to scale $\alpha \in \mathbb{R}$ which is called the point *weight*. It should be noted that the fact that the vector p is isotropic, i.e. $p^2 = 0$, implies that it is not invertible.

The geometric product of a number of invertible conformal vectors $s_i \in \mathbb{R}^{n+1,1}$, $i = 1, \dots, k$ results in a *versor* $s = s_1 \dots s_k$, an element of the Clifford group $\Gamma(\mathbb{R}^{n+1,1})$ that can be used to perform an orthogonal transformation on a geometric object $x \in \mathscr{G}(\mathbb{R}^{n+1,1})$ by the sandwiching geometric product $x' = sx\widehat{s}^{-1}$. Orthogonal transformations of elements of $\mathscr{G}(\mathbb{R}^{n+1,n})$ can be interpreted as conformal transformations of objects in $\mathbb{R}^n$.

It is also possible to represent hyperspheres, geometric objects and versors by 2×2-matrices with entries from $\mathscr{G}(\mathbb{R}^n)$, the geometric algebra over the n-dimensional Euclidean base space [1, 4]. For example, we are able to represent the isotropic elements of the null basis by

$$n_o \simeq \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \tag{6}$$

$$n_\infty \simeq \begin{pmatrix} 0 & -2 \\ 0 & 0 \end{pmatrix}, \tag{7}$$

the Euclidean elements by

$$\mathbf{e}_i \simeq \begin{pmatrix} \mathbf{e}_i & 0 \\ 0 & -\mathbf{e}_i \end{pmatrix}, \quad i = 1, \dots, n. \tag{8}$$

A conformal point at Euclidean position $\mathbf{p}$ is represented by

$$p \simeq P = \begin{pmatrix} \mathbf{p} & -\mathbf{p}^2 \\ 1 & -\mathbf{p} \end{pmatrix}. \tag{9}$$

The matrix representing an object from $\mathscr{G}(\mathbb{R}^{n+1,1})$ - called a *conformal object* from now on - is only defined up to scale. If we are not interested in that scale, i.e. the object's weight, we will often use $\simeq$ to suppress the weight and keep the matrices simple and recognizable.

The geometric product between conformal objects is now implemented by matrix multiplication between the matrix representations, where the (non-commutative) geometric product of $\mathscr{G}(\mathbb{R}^n)$ is employed between entries. Finally, the inner and outer product arise from their definitions as symmetric and anti-symmetric part of the geometric product. Let $v \in \mathbb{R}^{n+1,1}$, $w \in \mathscr{G}^k(\mathbb{R}^{n+1,1})$. Then

$$v \cdot w = \frac{1}{2}(vw - \widehat{w}v), \tag{10}$$

$$v \wedge w = \frac{1}{2}(vw + \widehat{w}v). \tag{11}$$

With this we can define a *conformal tangent* t anchored at a conformal point p at Euclidean position $\mathbf{p}$

and pointing in the direction of a Euclidean vector $\mathbf{t}$. In terms of CGA this tangent is a null blade that can be written in the form [3]

$$t = -p \cdot (p \wedge \mathbf{t} \wedge n_\infty)\,. \tag{12}$$

Representing the participating (conformal) vectors by matrices and carrying out the involved products using (10) and (11), we find the following matrix representation for the conformal tangent t.

$$t \;\simeq\; T = \begin{pmatrix} \mathbf{pt} & -\mathbf{ptp} \\ \mathbf{t} & -\mathbf{tp} \end{pmatrix} \tag{13}$$

The matrix representations of conformal vectors have to obey the conditions of the chosen vector basis. If we turn our attention to the matrix representations of versors, we find that – by their construction as the geometric product of a number of invertible conformal vectors – additional constraints arise. In general, we will call a matrix $S = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ with entries $a,b,c,d \in \mathcal{G}(\mathbb{R}^n)$ a *Vahlen matrix*, if it fulfills the conditions [4]

$$a,b,c,d \in \Gamma(\mathbb{R}^n) \cup \{0\}\,, \tag{14}$$

$$a\widehat{\overline{b}}, \widehat{\overline{b}}d, d\widehat{\overline{c}}, \widehat{\overline{c}}a \in \mathbb{R}^n \quad \text{and} \tag{15}$$

$$\Delta(S) = a\widehat{\overline{d}} - b\widehat{\overline{c}} \in \mathbb{R} \setminus \{0\}\,. \tag{16}$$

The concepts of grade involution and Clifford conjugation extend to Vahlen matrices as follows.

$$\widehat{S} = \begin{pmatrix} \widehat{a} & -\widehat{b} \\ -\widehat{c} & \widehat{d} \end{pmatrix} \tag{17}$$

$$\overline{S} = \begin{pmatrix} \widehat{\overline{d}} & -\widehat{\overline{b}} \\ -\widehat{\overline{c}} & \widehat{\overline{a}} \end{pmatrix} \tag{18}$$

A Vahlen matrix is always invertible with

$$S^{-1} = \frac{1}{\Delta(S)}\overline{S}. \tag{19}$$

Notice that the matrix representations given so far are not strictly Vahlen matrices and not necessarily invertible, since they fail condition (16).

Now let us assume a Vahlen matrix of the form

$$S = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{20}$$

Given a conformal object $X \in \mathcal{G}(\mathbb{R}^{n+1,1})$ in its matrix representation, Vahlen matrix S acts on it via the *twisted adjoint action*

$$\begin{aligned} X' &= SX\widehat{S}^{-1} \\ &= \frac{1}{\Delta(S)} SX\widehat{\overline{S}} \end{aligned} \tag{21}$$

Spelling out the action of a Vahlen matrix on a conformal tangent and a conformal point, respectively, we find that

$$ST\widehat{S}^{-1} = \frac{1}{\Delta(S)} \begin{pmatrix} \mathbf{p}'\mathbf{t}' & -\mathbf{p}'\mathbf{t}'\mathbf{p}' \\ \mathbf{t}' & -\mathbf{t}'\mathbf{p}' \end{pmatrix}, \tag{22}$$

$$SP\widehat{S}^{-1} = \frac{(c\mathbf{p}+d)\overline{(c\mathbf{p}+d)}}{\Delta(S)} \begin{pmatrix} \mathbf{p}' & -\mathbf{p}'^2 \\ 1 & -\mathbf{p}' \end{pmatrix} \tag{23}$$

where

$$\mathbf{t}' = (c\mathbf{p}+d)\,\mathbf{t}\,\overline{(c\mathbf{p}+d)} \quad \text{and} \tag{24}$$

$$\mathbf{p}' = (a\mathbf{p}+b)(c\mathbf{p}+d)^{-1}, \tag{25}$$

if $(c\mathbf{p}+d) \neq 0$, i.e. $(c\mathbf{p}+d)^{-1}$ exists. Otherwise we find that

$$ST\widehat{S}^{-1} \simeq \frac{1}{\Delta(S)} \begin{pmatrix} 0 & -2\mathbf{t}' \\ 0 & 0 \end{pmatrix}, \tag{26}$$

$$SP\widehat{S}^{-1} \simeq \begin{pmatrix} 0 & -2 \\ 0 & 0 \end{pmatrix}, \tag{27}$$

where

$$\mathbf{t}' = (a\mathbf{p}+b)\,\mathbf{t}\,\overline{(a\mathbf{p}+b)} \quad \text{and} \tag{28}$$

$$(c\mathbf{p}+d) = 0. \tag{29}$$

(26) and (27) represent a conformal tangent anchored at n_∞ (also called a *free vector* in [3]) and the conformal point at infinity n_∞, respectively.

Representing an even conformal versor by a Vahlen matrix helps both in design and interpretation of the transformation performed. Some basic transformations in Euclidean space are represented as follows.

- Translation along a Euclidean vector $\mathbf{t}$

$$S = \begin{pmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{pmatrix} \tag{30}$$

- Transversion parametrized by Euclidean vector $\mathbf{v}$

$$S = \begin{pmatrix} 1 & 0 \\ \mathbf{v} & 1 \end{pmatrix} \tag{31}$$

- Rotation by Euclidean rotor R

$$S = \begin{pmatrix} R & 0 \\ 0 & R \end{pmatrix} \tag{32}$$

- Uniform scaling by factor $\exp(\gamma)$, $\gamma \in \mathbb{R}$

$$S = \begin{pmatrix} \exp(\gamma/2) & 0 \\ 0 & \exp(-\gamma/2) \end{pmatrix} \tag{33}$$

Note that transversion (31) and scaling (33) are able to change the weight of transformed points.

These simple structures make it easy to predict the interaction between basic transformations or analyze a composite transformation. Also, this representation facilitates the interpretation of the algebraic result of a transformation in terms of its effects on the Euclidean parts of a conformal object. Note that these simple transformations trivially fulfill the Vahlen conditions (14), (15) and (16).

Because one often works with the null basis representation of CGA, it is useful to have a conversion from the matrix representation available.

$$\begin{aligned} s &= \frac{1}{2}(A+\widehat{D})+\frac{1}{2}(A-\widehat{D})E+\widehat{C}n_o-\frac{1}{2}Bn_\infty \\ &\in \mathscr{G}(\mathbb{R}^{n+1,1}) . \end{aligned} \tag{34}$$

3 POINT AND TANGENT TRANSFER

Let us assume that we have two sets of conformal objects $\{x_i\}$, $\{x_i'\}$, $i=1,\ldots,N$ with element-wise correspondences, and that we know that they are related by a conformal transformation. Our goal is to determine that conformal transformation by determining the (even) versor s that performs this transformation by $sx_is^{-1}=x_i'$ for all $i=1,\ldots,N$.

3.1 Even and Odd Versors

So far we have used the term “conformal transformation” rather loosely to mean a transformation that locally preserves the magnitude of angles and only implied that the angles’ orientation (or handedness) is also preserved. In fact, conformal geometric algebra provides means to represent both, transformations that locally preserve angle magnitudes as well as orientations and transformations that locally preserve angle magnitudes but *reverse* all orientations.

An even versor in CGA is the geometric product of an even number of invertible conformal vectors. It induces a conformal transformation, which preserves angles locally. As such it is opposed to an odd versor, which is the geometric product of an odd number of vectors and induces a locally angle-reversing *anti-conformal* transformation. In the Vahlen matrix representing an even (odd) versor, the diagonal elements are the geometric product of an even (odd) number of Euclidean vectors, while the off-diagonal entries are the geometric product of an odd (even) number of Euclidean vectors and its determinant (16) is positive.

It has been shown [5] that every even versor in the geometric algebra over a Euclidean or a Minkowski space can be written as the exponential of a *bivector* $s=\exp(b)$, $\langle b\rangle_2=b$. Conversely, the exponential of any bivector results in an even versor. Note that in general b is not a 2-blade, i.e. $b\notin\mathscr{G}^2(\mathbb{R}^{n+1,1})$, but may be written as a sum of 2-blades. This observation provides us with a counting argument for the number of degrees of freedom of an even versor. It is equal to the number of degrees of freedom of a bivector in the geometric algebra over the given space. In case of CGA we have

$$\#\text{DOF of an even versor} = \begin{pmatrix} n+2 \\ 2 \end{pmatrix} . \tag{35}$$

For now, we will restrict our considerations to the specific case of $n=3$. Then (35) amounts to

$$\#\text{DOF of an even versor} = \begin{pmatrix} 5 \\ 2 \end{pmatrix} = \frac{5!}{2!\,3!} = 10 , \tag{36}$$

which limits our choice for the object correspondences $\{x_i\}\leftrightarrow\{x_i'\}$ that we observe. To explain our basic method we take a well chosen minimal set of data as follows.

3.2 Minimal Data

Let us begin with a vector frame $\{\mathbf{t}_i\}$, $i=1,\ldots,3$, which, without loss of generality, we can assume to be orthonormal, because every non-orthogonal frame can be orthonormalized using Gram-Schmidt orthogonalization in the Euclidean space $\mathbb{R}^3$. By taking this frame and anchoring it at a given point, the frame’s Euclidean location in space clearly provides us with three degrees of freedom. The first direction vector provides two more as, since it is normalized, only its direction matters but not its scale. The second direction vector provides only one degree of freedom, because it has to be orthogonal to the first and is normalized, too. The third direction vector is completely determined by the former two because of the orthonormalization assumption we made. The corresponding vector frame $\{\mathbf{t}_i'\}$ that $\{\mathbf{t}_i\}$ is mapped to by the conformal transformation is orthogonal, because conformal transformations preserve angles. But the vectors may be related by a common scaling factor, since conformal transformations allow for uniform scaling, thus providing one more degree of freedom. So far we have accounted for $3+2+1+1=7$ degrees of freedom, which leaves us three degrees of freedom short of specifying a conformal transformation. They can be provided by observing the transformation of an additional point in Euclidean 3-space.

In summary, we assume a frame made up of three conformal tangents t_1,t_2,t_3 anchored at a common conformal point p_1 and an additional conformal point p_2, all being mapped to another frame t_1',t_2',t_3' anchored at a common point q_1 and an additional point q_2. In CGA it is quite hard to solve for the even versor mapping these two sets to each other, because of the non-commutative geometric product and interactions between the different multivectors involved in intermediate stages of the mapping.

3.3 Approach to Solution

In order to reduce the complexity and dimensionality of the equations to solve, we employ two main ideas.

Firstly, we turn to the matrix representations of the conformal objects. This gives us

$$\begin{aligned} x_i &= T_i \\ &= \begin{pmatrix} \mathbf{p}_1\mathbf{t}_i & -\mathbf{p}_1\mathbf{t}_i\mathbf{p}_1 \\ \mathbf{t}_i & -\mathbf{t}_i\mathbf{p}_1 \end{pmatrix}, \quad i=1,2,3 \quad (37) \\ x_4 &= P_2 \\ &= \begin{pmatrix} \mathbf{p}_2 & -\mathbf{p}_2^2 \\ 1 & -\mathbf{p}_2 \end{pmatrix}, \quad (38) \end{aligned}$$

for the "original" set and analogous expressions for the "image" set $\{x_i'\}$.

Secondly, we will introduce a set of *transfer objects* $\{y_i\}$, which have a very simple matrix representation. This idea is similar to the "projective frame" technique using homogeneous coordinates to calculate *Möbius transformations* in the 2D (complex) plane. In particular, we introduce

$$\begin{aligned} y_i &= T_i^0 \\ &= \begin{pmatrix} 0 & 0 \\ \mathbf{t}_i^0 & 0 \end{pmatrix}, \quad i=1,2,3 \quad (39) \\ y_4 &= P^\infty = n_\infty \\ &= \begin{pmatrix} 0 & -2 \\ 0 & 0 \end{pmatrix}, \quad (40) \end{aligned}$$

i.e. a frame at the origin and the point at infinity. Then we try to find the Vahlen matrices $S_j = \begin{pmatrix} a_j & b_j \\ c_j & d_j \end{pmatrix}$, $j = 1,2$, which take each of the sets $\{x_i\}$ and $\{x_i'\}$ of conformal objects to these transfer objects, respectively, i.e. $S_1 x_i \widehat{S_1}^{-1} = y_i$ and $S_2 x_i' \widehat{S_2}^{-1} = y_i$ for all $i = 1,\dots,4$. Finally, we find the conformal transformation that takes $\{x_i\}$ to $\{x_i'\}$ as $S = S_2^{-1}S_1$, such that

$$S x_i \widehat{S}^{-1} = x_i' \quad \text{for all } i = 1,\dots,4. \qquad (41)$$

3.4 Solution

Consider S_1 acting on (37) and (38) via (22) and (27), respectively. This takes all the tangents to the origin, i.e. $\mathbf{p}_1' = 0$, and the additional point to infinity.

By (25) and (29) we find that

$$\begin{aligned} a_1\mathbf{p}_1 + b_1 = 0, &\quad \textit{so} \quad b_1 = -a_1\mathbf{p}_1 \quad \text{and} \quad (42) \\ c_1\mathbf{p}_2 + d_1 = 0, &\quad \textit{so} \quad d_1 = -c_1\mathbf{p}_2\,. \quad (43) \end{aligned}$$

Next we ensure that S_1 is indeed a Vahlen matrix by asserting conditions (14), (15) and (16).

Remember (see beginning of section 3) that S has to represent an even versor. Therefore either S_1 and S_2 both represent even versors or both represent odd versors. Without loss of generality we will assume that both represent even versors. Consequently, a_1 has to be the geometric product of an even number of Euclidean vectors and c_1 must be the geometric product of an odd number of Euclidean vectors. By this, condition (14) is fulfilled.

We evaluate condition (16) to

$$\begin{aligned} \mathbb{R}\setminus 0 \ni \Delta(S_1) &= a_1\widehat{\overline{d_1}} - b_1\widehat{\overline{c_1}} \\ &= -a_1\widehat{\overline{c_1\mathbf{p}_2}} + a_1\mathbf{p}_1\widehat{\overline{c_1}} \\ &= a_1\mathbf{p}_1\widehat{\overline{c_1}} - a_1\widehat{\overline{\mathbf{p}_2}}\,\widehat{\overline{c_1}} \\ &= a_1(\mathbf{p}_1 - \mathbf{p}_2)\widehat{\overline{c_1}}, \qquad (44) \end{aligned}$$

so that

$$a_1^{-1} = \frac{1}{\Delta(S_1)}(\mathbf{p}_1 - \mathbf{p}_2)\widehat{\overline{c_1}}\,. \qquad (45)$$

This also fulfills condition (15) as we see when we write out the four individual equations.

So far, our Vahlen matrix S_1 is of the form

$$S_1 = \begin{pmatrix} a_1 & -a_1\mathbf{p}_1 \\ c_1 & -c_1\mathbf{p}_2 \end{pmatrix}, \qquad (46)$$

which, with (45), leaves the odd Euclidean versor c_1 as the only unknown. We can find a solution using (39) and (22), which claims that

$$\begin{aligned} \mathbf{t}_i^0 &= (c_1\mathbf{p}_1 + d_1)\,\mathbf{t}_i\,\overline{(c_1\mathbf{p}_1 + d_1)}, \quad i = 1,2,3 \\ &= (c_1\mathbf{p}_1 - c_1\mathbf{p}_2)\,\mathbf{t}_i\,\overline{(c_1\mathbf{p}_1 - c_1\mathbf{p}_2)} \\ &= c_1(\mathbf{p}_1 - \mathbf{p}_2)\,\mathbf{t}_i\,\overline{c_1(\mathbf{p}_1 - \mathbf{p}_2)}\,. \qquad (47) \end{aligned}$$

If we normalize the tangent vectors $\mathbf{t}_i = ||\mathbf{t}_i||\check{\mathbf{t}}_i$ and introduce the even Euclidean versor $c_1' = c_1(\mathbf{p}_1 - \mathbf{p}_2)$, we are allowed to write

$$\check{\mathbf{t}}_i^0 = \frac{||\mathbf{t}_i||}{||\mathbf{t}_i^0||}\, c_1'\check{\mathbf{t}}_i c_1'^{-1}, \quad i = 1,2,3 \qquad (48)$$

and find that we have reduced our problem to finding an even Euclidean rotor c_1' which rotates an orthonormal Euclidean vector frame $\{\check{\mathbf{t}}_1, \check{\mathbf{t}}_2, \check{\mathbf{t}}_3\}$ onto another one $\{\check{\mathbf{t}}_1^0, \check{\mathbf{t}}_2^0, \check{\mathbf{t}}_3^0\}$. In [6] we find a solution for this even versor up to a scaling factor. We let

$$c_1' = 1 + \sum_{i=1}^{3} \check{\mathbf{t}}_i^0 \check{\mathbf{t}}_i\,, \qquad (49)$$

and calculate

$$c_1 = \sqrt{\frac{||\mathbf{t}_i^0||}{||\mathbf{t}_i||\,c_1'\widehat{\overline{c_1'}}}}\; c_1'(\mathbf{p}_1 - \mathbf{p}_2)^{-1}\,. \qquad (50)$$

Now, substituting (50) and (45) back into (46) gives us a full closed-form solution to S_1. A solution for S_2 can be found in a perfectly analogous way and we find the Vahlen matrix $S = S_2^{-1}S_1$ by straightforward application of (19).

The Vahlen matrix S representing an even versor can be converted back into a multivector expression in terms of the null basis of CGA by interpretation formula (34).

4 GENERALIZATION AND LIMITATIONS

In this section we generalize our method to odd versors and arbitrary dimensions. We analyze the important special case of rigid body motions and reflect on the general applicability and limitations of our solution.

4.1 Even vs. Odd Versors

Throughout the procedure we have assumed that the corresponding data is related by a conformal transformation, which can be represented by an even versor. If the two configurations are related by a locally angle reversing anti-conformal transformation, the method has to be modified slightly.

In this case the handedness of only one of the localized vector frames will match the handedness of our transfer frame. We can compute handedness for example by the sign of the expression $\mathbf{t}_1 \wedge \mathbf{t}_2 \wedge \mathbf{t}_3$. On the *other* configuration (without loss of generality let this be $\{x'_i\}$) we introduce a reflection in an arbitrary but fixed hypersphere represented by a conformal vector r and obtain $\{x''_i\} = -r\{x'_i\}r^{-1}$. Now we compute the even conformal versor S' relating $\{x_i\}$ and $\{x''_i\}$ and find the desired odd conformal versor relating $\{x_i\}$ and $\{x'_i\}$ by undoing the reflection introduced before:

$$S = -r^{-1}S'r. \tag{51}$$

4.2 Closed Form in nD

Even though we have derived our formulas for the example of $n = 3$, it should be noted that, up to (49), we have not made an explicit assumption about the dimensionality n of the represented base space. We find that, in general, a localized orthonormal Euclidean vector frame consisting of n direction vectors provides $(n-1)+(n-2)+\cdots+1+0$ degrees of freedom, their common scale provides one more, their location n more and an additional Euclidean point provides an additional n degrees of freedom, making for a sum of

$$\begin{aligned}\#\text{DOF} &= (n-1)+\cdots+1+1+n+n \\ &= 1+n+\frac{n(n+1)}{2} \\ &= \frac{(n+2)(n+1)}{2} \\ &= \begin{pmatrix} n+2 \\ 2 \end{pmatrix}\end{aligned} \tag{52}$$

in accordance with (35), which suggests that our method would provide a solution for arbitrary n. The limitation we face, however, is that formula (49) only works in $n = 3$ dimensions and would have to be generalized to arbitrary n in order to apply our method. A closed form solution to this problem is given in [2].

Another observation is that our source [6] for formula (49) does not require an orthonormal frame. By substituting the Euclidean transfer frame with its *reciprocal frame*, we find a solution for an arbitrary Euclidean frame. Since conformal transformations preserve angles locally, however, the mutual angles between the vectors of the three frames $\{\mathbf{t}_i\}$, $\{\mathbf{t}'_i\}$ and $\{\mathbf{t}^0_i\}$ (in particular, their handedness; see section 4.1) have to correspond. Also, one has to make sure that the frames are pairwise related by one common scale. For these reasons, explicit orthonormalization can be advantageous, depending on the way the data is acquired.

Finally, there exist certain singular configurations which do not yield a closed form solution. For example, if $\{\mathbf{t}_i\}$ and $\{\mathbf{t}'_i\}$ are related by a rotation through π radians, (49) will result in zero. The reason for that is that it is impossible to determine a unique plane of rotation. Rather, a one parameter family of possible rotation planes exists. Note that this ambiguity is inherent in the geometry of the problem and cannot be resolved analytically. The same problem occurs in higher dimensions [2].

4.3 Special Case: Rigid Body Motions

An important special case of conformal transformations are rigid body motions or Euclidean transformations, which preserve angles *globally*. If it is known beforehand that the transformation we seek is a rigid body motion, the equations simplify significantly.

A convenient way to incorporate this additional information would be to specify the point at infinity as being preserved, e.g. $P_2 = P'_2 = P^\infty$. However, our method depends on the Euclidean position vectors $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}'_1$, $\mathbf{p}'_2$ of the given points, respectively. The point at infinity cannot be specified explicitly in terms of Euclidean position vectors, breaking down equations (23) and (25). We must replace equations (42), (43), (45) and (50) by the following considerations.

The action of a Vahlen matrix S on the point at infinity is given by

$$\begin{aligned} SP^\infty\widehat{S}^{-1} &\simeq \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} 0 & -2 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} \overline{d} & \overline{b} \\ \overline{c} & \overline{a} \end{pmatrix} \\ &= \begin{pmatrix} -2a\overline{c} & -2a\overline{a} \\ -2c\overline{c} & -2c\overline{a} \end{pmatrix}. \end{aligned} \tag{53}$$

Thus, if we want the point at infinity to be preserved, our conditions for S_1 simplify to

$$c_1 = 0, \tag{54}$$

$$a_1 = \Delta(S)\widehat{\overline{d_1}}^{-1}, \tag{55}$$

$$b_1 = -a_1\mathbf{p}_1, \tag{56}$$

$$d_1\,\mathbf{t}_i\,\overline{d_1} = \mathbf{t}^0_i, \tag{57}$$

and analogously for S_2, which yields a solution for S using the above methods.

5 MINIMAL DATA FROM EXACT POINT CORRESPONDENCES

Note that a localized vector frame and an additional point are minimal data. That means that *every* such correspondence uniquely determines a conformal transformation. However, sometimes it may be inconvenient to give the correspondence data in this format, e.g. because of the way the data is acquired. In this section we will introduce a way to acquire a minimal set of correspondence data from point correspondences.

Several methods are being developed to determine a conformal versors from correspondence data. The approach in [2] can also determine a conformal versor from exact point correspondences, but only if the point weights are known. A general conformal transformation can change point weights, whereas rigid body motions (i.e. purely Euclidean transformations) preserve them. Our method here has the advantage of working with purely Euclidean geometric information, i.e. point locations in Euclidean space.

A crucial prerequisite is that the points are indeed related by a conformal transformation, as the counting argument suggests. For example, in three dimensions a single point correspondence provides three degrees of freedom. Therefore, three point correspondences would provide only 9 DOF leaving us one degree of freedom short of specifying a conformal transformation. On the other hand, four point correspondences provide 12 DOF, thus overdetermining the problem.

If we assume the existence of a conformal transformation, we can use the properties of conformal transformations and the representational power of CGA to find a closed form solution to (41). Again, we restrict our explanations to three dimensions, but the principles generalize to arbitrary dimensions.

First of all, we observe that conformal transformations preserve hyperspheres (depending on their dimensionality these can be point pairs, circles, spheres etc.), i.e. hyperspheres are mapped to hyperspheres. Secondly, conformal transformations preserve angles locally, i.e. if two hyperspheres intersect at a certain angle, their images intersect at the same angle. Therefore, in three dimensions, two circles intersecting in a point at Euclidean position $\mathbf{p}$ are enough to specify a vector frame localized at $\mathbf{p}$ up to scale. Two of the vectors are given as the tangent directions of the circles in $\mathbf{p}$ and the third one can be determined as being orthogonal to both.

Three points uniquely determine a circle. In CGA the circle can be directly represented as the outer product of three conformal points. In order to specify two independent circles we need at least four points $\{p_1, p_2, p_3, p_4\}$. Without loss of generality we pick the circles

$$c_1 = p_1 \wedge p_2 \wedge p_3 \quad \text{and} \tag{58}$$

$$c_2 = p_1 \wedge p_2 \wedge p_4 \tag{59}$$

intersecting in p_1. The tangents to these circles in p_1 are given by

$$t_1 = p_1 \cdot c_1 \quad \text{and} \tag{60}$$

$$t_2 = p_1 \cdot c_2, \tag{61}$$

their Euclidean directions can be recovered and the local frame specified by

$$\mathbf{t}_1 = -n_\infty \cdot t_1, \tag{62}$$

$$\mathbf{t}_2 = -n_\infty \cdot t_2 \quad \text{and} \tag{63}$$

$$\mathbf{t}_3 = -(\mathbf{t}_1 \wedge \mathbf{t}_2) I_3, \tag{64}$$

where I_3 denotes the three-dimensional pseudoscalar $I_3 = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$.

Due to the preservation properties of conformal transformations, the corresponding vector frame could be determined perfectly analogously by replacing the conformal points by their images $\{p'_1, p'_2, p'_3, p'_4\}$ in equations (59) through (64).

However, for our closed form solution to work, the respective angles between frame vectors not only have to be preserved, they also have to be *known*, because in an intermediate step we match the frames with the transfer frame. To ensure that the angles in the localized frames and in the transfer frame match, we prefer to employ an orthonormalization procedure [5] very similar to Gram-Schmidt orthonormalization. This enables us to pick a cartesian frame $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ as the transfer frame. With this and an additional point correspondence (without loss of generality we can pick any point other than p_1, say, p_2) we have all the information to compute the versor *s up to the scale parameter σ*.

Because of the orthonormalization of both localized frames we cannot recover the scaling factor. Also, this scaling factor cannot be recovered by the properties of the point correspondences directly, because we assumed that we have no information about their weights. Fortunately the scaling factor is just one scalar parameter and it can be found as follows.

If the original frame and its image are related to the transfer frame by scaling factors σ_1 and σ_2, respectively, they are related to each other by scaling factor $\sigma = \sigma_1 / \sigma_2$. Without loss of generality we assume $\sigma_1 = \sigma$ and $\sigma_2 = 1$. Applying the pure scaling transformation (33), we find that we can replace (46) by

$$S_1 = \begin{pmatrix} a_1 & b_1 \\ \sigma c_1 & \sigma d_1 \end{pmatrix} \tag{65}$$

and track the scaling factor through the whole process. It turns out that the final Vahlen matrix takes on the form

$$\begin{aligned} S &= S_2^{-1} S_1 \\ &\simeq \begin{pmatrix} \widehat{\overline{d_2}} a_1 - \sigma \widehat{\overline{b_2}} c_1 & \widehat{\overline{d_2}} b_1 - \sigma \widehat{\overline{b_2}} d_1 \\ -\widehat{\overline{c_2}} a_1 + \sigma \widehat{\overline{a_2}} c_1 & -\widehat{\overline{c_2}} b_1 + \sigma \widehat{\overline{a_2}} d_1 \end{pmatrix} \end{aligned} \tag{66}$$

its effect on the Euclidean position $\mathbf{p}$ of a point being

$$\begin{aligned}\mathbf{p}' \;=\;& ((\widehat{\overline{d_2}}a_1 - \sigma\widehat{\overline{b_2}}c_1)\mathbf{p} + \widehat{\overline{d_2}}b_1 - \sigma\widehat{\overline{b_2}}d_1) \\ & ((\sigma\widehat{\overline{a_2}}c_1 - \widehat{\overline{c_2}}a_1)\mathbf{p} + \sigma\widehat{\overline{a_2}}d_1 - \widehat{\overline{c_2}}b_1)^{-1}, \quad (67)\end{aligned}$$

which can be solved for σ giving

$$\begin{aligned}\sigma \;=\;& (\mathbf{p}'\widehat{\overline{c_2}}a_1\mathbf{p} + \mathbf{p}'\widehat{\overline{c_2}}b_1 + \widehat{\overline{d_2}}a_1\mathbf{p} + \widehat{\overline{d_2}}b_1) \\ & (\mathbf{p}'\widehat{\overline{a_2}}c_1\mathbf{p} + \mathbf{p}'\widehat{\overline{a_2}}d_1 + \widehat{\overline{b_2}}c_1\mathbf{p} + \widehat{\overline{b_2}}d_1)^{-1}. \quad (68)\end{aligned}$$

In order to find the correct scaling parameter, we determine the one parameter family of Vahlen matrices $S(\sigma)$ by (66) and pick one Vahlen matrix with a specific scale parameter, e.g. set $\sigma = 1$. Note that by design the points p_1 and p_2 will be aligned to p'_1 and p'_2 by the Vahlen matrix, no matter which value we choose for σ. We calculate the Euclidean position of an additional transformed point (without loss of generality we choose p_3) $P'_3 = S(1)P_3\widehat{S(1)}^{-1}$ and substitute the result into (68) yielding σ.

6 CONCLUSION

We have derived a closed form solution to a conformal versor given exact correspondences between two sets of data. We have shown that a minimal set of data needed is a localized Euclidean vector frame and an additional point in Euclidean space. It turns out that this provides exactly the number of degrees of freedom necessary to completely specify a conformal (resp. anti-conformal), i.e. locally angle preserving (resp. angle reversing), transformation.

In order to derive this solution we have made use of the representation of the conformal model of geometric algebra by matrices. This representation reduces the problem of calculating a general even conformal versor to that of finding an even Euclidean rotor, thereby reducing the dimensionality of the problem.

Moreover, the method works with purely Euclidean geometric information on point locations and direction vectors. As opposed to other methods [2, 5, 7] knowledge of the weights of conformal points is not required.

A counting argument suggests generalizability of the closed form solution to arbitrary dimensions of the represented base space using the closed form generalization of the rotary frame matching (49) in [2].

Interesting subjects for further research include approximate solutions under non-exact correspondences, optimization of those solutions under certain aspects, e.g. minimization of Euclidean distances, as well as propagation of errors or uncertainty in the given data.

REFERENCES

[1] R. Abłamowicz, P. Lounesto, and J. Parra. *Clifford Algebras with Numeric and Symbolic Computations*. Birkhäuser, 1996.

[2] L. Dorst. Determining an Even Versor in n-D Geometric Algebra from the Known Transformation of n Vectors. In *GraVisMa*, 2009.

[3] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science*. Morgan Kaufmann, 2007.

[4] U. Hertrich-Jeromin. *Introduction to Möbius Differential Geometry*. Cambridge University Press, 2003.

[5] D. Hestenes and G. Sobczyk. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. D. Reidel, Dordrecht, 1984.

[6] A.N. Lasenby and C. J. L. Doran. New geometric methods for computer vision: an application to structure and motion estimation. *International Journal of Computer Vision*, 26:191–213, 1998.

[7] C. Perwass. *Geometric Algebra with Applications in Engineering*. Springer, 2009.

Determining a Versor in n-D Geometric Algebra from the Known Transformation of n Vectors

Leo Dorst
University of Amsterdam, The Netherlands
L.Dorst@uva.nl

ABSTRACT

Suppose we only know of some elements in a geometric algebra how a versor has transformed them, can we then reconstruct the unknown versor V?
We present an $O(2^n)$ method that works in n-D geometric algebra for n exact vector correspondences. This makes it usable for determining, for instance, a Euclidean rigid body motion in n-D from a frame correspondence providing the required $n+2$ conformal vectors as: the frame location, the n axis directions, and the invariance of the point at infinity. The method can only determine a full conformal transformation if the weights of the transformed entities are also observed.

Keywords: geometric algebra, versor, correspondence data, conformal model, Euclidean motion, rigid body motion

1 CORRESPONDENCE DATA GIVES PROJECTED VERSOR INFORMATION

In geometric algebra, common transformations of elements in n-D can be represented as orthogonal transformations in a well-chosen m-D representational space. These orthogonal transformations are then represented by versors, to make them applicable to any element of the algebra (see e.g. [4, 3]). Examples are rotations in Euclidean $\mathbb{R}^n$ represented by rotors (with quaternions for $\mathbb{R}^3$ as a special case); and conformal transformations of $\mathbb{R}^n$ represented as versors in the conformal model $\mathbb{R}^{n+1,1}$ (with Euclidean rigid body motions as a special case).

We cannot observe versors directly, but we can look at the motions of geometrical elements. Clearly, we would like to have a technique that uses these observations and finds the versor that caused them. Some techniques for this exist [4, 6], but their applicability to n-D has not been shown; we treat them below in their appropriate context.

Though a truly practical method should be able to handle noise (and moreover preferably optimally), in this paper we consider exact data only. And to make the problem more tractable, we first limit this data to be direct observations of *vectors* in the geometric algebra of the versor (rather than higher grade elements, or locations of geometrical elements in the space modelled by the algebra).

So let x be a vector, and the primed symbol x' the corresponding transformed vector under the action of a versor V. Of course x' should be equal to $\widehat{V}xV^{-1}$ (with $\widehat{\cdot}$ denoting the grade involution), but we do not know V. Still, observing that x transforms to x' gives us some information on V. We can see this by rewriting the *chord* $x'-x$:

$$\tfrac{1}{2}(x-x') = \tfrac{1}{2}(x-\widehat{V}x/V) = \tfrac{1}{2}(xV-\widehat{V}x)V^{-1} = (x\rfloor V)/V. \quad (1)$$

(note that the final rewriting as a contraction uses the fact that x is of grade 1). We recognize the projection formula from geometric algebra (though this is usually only applied to the case when V is a blade representing the subspace of projection). The chord of x, which is measurable, therefore provides the projected component of x onto the unknown versor V. Intuitively, we would expect to be able to determine the versor V from having enough projections available. This is indeed possible, in a structured manner, and this paper shows how.

We know from the theory of reciprocal frames [4, 3] that a general multivector V of the geometric algebra of an n-dimensional metric vector space can be expressed using its components under the scalar product on a complete basis $\{e^J\}$ for the algebra,

$$V = \sum_J e^J\,(e_J * V).$$

Probing the versor with a vector leads to eq.(1), which does not contain a scalar product (which acts as a grade selector) but a contraction (which does not). We can use the measurements to form the quantity

$$D_1 = \sum_{\text{basis vectors } x} x^r\,(x-x')/2 = \sum_{\text{basis vectors } x} x^r(x\rfloor V)/V$$

(using $\cdot^r$ as a notation for the reciprocal vector within the chosen basis), but this does now not give V immediately. Instead, it may be derived that:

$$\begin{aligned}\sum_{\text{basis vectors } x} & x^r\,(x\rfloor V)/V = \\ &= \big(\tbinom{0}{1}V_0 + \tbinom{1}{1}V_1 + \tbinom{2}{1}V_2 + \cdots\big)/V \\ &= \big(V_1 + 2V_2 + 3V_3 + 4V_4 + \cdots\big)/V, \qquad (2)\end{aligned}$$

where $V_i \equiv \langle V \rangle_i$, the i-th grade part of V (for a derivation, see [3], pg. 258, or eq.(11) for the more general case).

Since versors are either fully odd or fully even, two cases may be distinguished. For ease in explanation, we first focus on even versors, and start with those having a highest nonzero grade of 2.

2 HIGHEST EVEN VERSOR GRADE 2

If we are trying to determine an even versor that has highest grade 2, the quantity D_1 is sufficient for determining V_2/V (namely, as $\frac{1}{2}D_1$ by eq.(2)). That quantity can in turn be related to V/V_0 by $V_2/V = (V - V_0)/V = 1 - V_0/V$. Working out this special case therefore gives a way to compute the original even versor (modulo a scalar factor):

$$\begin{aligned} V/V_0 &= \Big(1 - \tfrac{1}{2}\sum_{\text{basis } x} x^r\,(x \rfloor V)/V\Big)^{-1} \\ &= \Big(1 - \tfrac{1}{4}\sum_{\text{basis } x} x^r\,(x - x')\Big)^{-1} \\ &\propto \Big(4 - n + \sum_{\text{basis } x} x^r x'\Big)^{-1} \\ &\propto 4 - n + \sum_{\text{basis } x} x' x^r. \end{aligned} \tag{3}$$

In [5], the formula for the case $n = 3$ in $\mathbb{E}^3$ is given, which then determines a rotation rotor from a frame before and after rotation (in an effectively equivalent form that puts the reciprocal on x'). We now recognize that we can use eq.(3) in other spaces and metrics, such as in the $(m+2)$-D conformal geometric algebra (i.e., in $\mathbb{R}^{m+1,1}$) used as a model of $\mathbb{E}^m$.

> For instance, we may use eq.(3) to determine a translation versor in $\mathbb{R}^{m+1,1}$ (with null basis $\{n_o, n_\infty\}$ for $\mathbb{R}^{1,1}$ and an orthonormal basis $\{\mathbf{e}_i\}$ for the Euclidean subspace) from the data: $n_o \mapsto n_o + \mathbf{t} + \frac{1}{2}\mathbf{t}^2 n_\infty$ and $\mathbf{e}_i \mapsto \mathbf{e}_i + (\mathbf{e}_i \cdot \mathbf{t})\, n_\infty$ and $n_\infty \mapsto n_\infty$ (showing where the origin goes, that orientations do not change, and that the point at infinity is preserved). Using $n_o{}^r = -n_\infty$, $n_\infty{}^r = -n_o$, and $\mathbf{e}_i{}^r = \mathbf{e}_i$, we get:
>
> $$\begin{aligned} V/V_0 &\propto (4 - m - 2) + \sum_{i=1}^{m} (\mathbf{e}_i + (\mathbf{e}_i \cdot \mathbf{t})\, n_\infty)\, \mathbf{e}_i \\ &\quad - (n_o + \mathbf{t} + \tfrac{1}{2}\mathbf{t}^2 n_\infty)\, n_\infty - n_\infty n_o \\ &= (2 - m) + (m + n_\infty \mathbf{t}) + 2 - \mathbf{t} n_\infty \\ &\propto 1 - \mathbf{t} n_\infty/2, \end{aligned}$$
>
> which is the correct answer.

However, general rigid body motion versors in the conformal model of $\mathbb{E}^3$ may also contain grade 4 parts, so more is needed even in such a low-dimensional (but important) case.

3 EVEN VERSOR, HIGHER GRADES

If the parts of the even versor V of grade higher than 2 are not zero, the vector-based method of eq.(3) is not sufficient to determine the full V/V_0, since by eq.(2) it then only determines the combination $D_1 = (2V_2 + 4V_4 + \cdots)/V$ from the vector correspondences. To obtain independent information on the other parts, we need to probe with higher order elements. For instance, probing with all 2-blades from a 2-blade basis would give a quantity we label D_2:

$$\begin{aligned} D_2 &\equiv \sum_{\text{basis 2-blades } X} X^r\,(X \rfloor V)/V \\ &= \big(\tbinom{0}{2} V_0 + \tbinom{2}{2} V_2 + \tbinom{4}{2} V_4 + \cdots\big)/V \\ &= \big(V_2 + 6V_4 + \cdots\big)/V, \end{aligned} \tag{4}$$

(proof of this identity below in eq.(11)), and therefore provides an independent equation on the grade parts $V_i = \langle V \rangle_i$ of V. But if we are to use this in practice, we should make sure that the sum on the left hand side can be performed on the measured data, i.e., that $(X \rfloor V)/V$ is measurable for all 2-blades X in some basis for the 2-blades of the geometric algebra of the n-D vector space.

In some setups, one may be able to measure the 2-blades directly; but the most obvious way to measure the transformation results on the elements in a basis of 2-blades is: from the n vector correspondences already measured. A simple expansion shows how a term involving the 2-blade $(y \wedge x)$ can be constructed from the observed vector correspondences $x \mapsto x'$ and $y \mapsto y'$:

$$\begin{aligned} \big((y \wedge x) \rfloor V\big)/V &= \big(y \rfloor (x \rfloor V)\big)/V \\ &= \tfrac{1}{2}\big(y \rfloor (xV - Vx)\big)/V \\ &= \tfrac{1}{4}\big(yxV - yVx + xVy - Vxy\big)\big)/V \\ &= \tfrac{1}{4}(yx - yx' + xy' - x'y'). \end{aligned} \tag{5}$$

This outcome can be determined completely from the known correspondences $x \mapsto x'$ and $y \mapsto y'$, and is therefore measurable from vector correspondences. From n independently measured vectors, we can then construct the $\binom{n}{2}$-dimensional basis of 2-blades. This in turn makes D_2 in eq.(4) computable from the data, and therefore gives us the quantity $(V_2 + 6V_4 + \cdots)/V$. Other independent equations for the other grades can be obtained in a similar manner, giving the general method summarized in Section 4.

As an example, if we now restrict ourselves to the important conformal model of 3D space, the vector dimension is $n = 5$, and the highest grade for an even versor is 4; so D_1 and D_2 should suffice to determine V. To make this explict, we have:

$$\begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 1 & 6 \end{bmatrix} \begin{bmatrix} V_2/V \\ V_4/V \end{bmatrix}$$

so

$$\begin{bmatrix} V_2/V \\ V_4/V \end{bmatrix} = \begin{bmatrix} 6/8 & -4/8 \\ -1/8 & 2/8 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \end{bmatrix}.$$

Then the even versor W with no higher grade than 4 is determined from 5 vector correspondences as:

$$\begin{aligned} W \equiv V/V_0 &= (1 - V_2/V - V_4/V)^{-1} \\ &\propto (8 - 5D_1 + 2D_2)^{-1} \\ &\propto 8 - 5\widetilde{D}_1 + 2\widetilde{D}_2. \end{aligned} \tag{6}$$

If a normalized even versor V (a.k.a. a rotor) is required, this can be retrieved as $V = W/\sqrt{W\widetilde{W}}$.

There are alternative ways of computing the same versor. In $\mathbb{R}^{4,1}$ the highest grade part of an even versor is of grade 4. We can define

$$D_4 \equiv \sum_{\text{basis 4-blades } X} X^r\,(X \rfloor V)/V = V_4/V \quad (\text{for } n < 6). \tag{7}$$

We might be able to measure where 5 independent spheres or planes go, thus directly providing transformation results on a complete basis of 4-blades. Or we can determine $(X\rfloor V)/V$ for all X in a complete basis of 4-blades from vector correspondence data, by a simple expansion similar to eq.(5):

$$\begin{aligned} &\big((d \wedge c \wedge b \wedge a)\rfloor V\big)/V = \\ &= \tfrac{1}{16}\,(dcba - dcba' + dcab' - dca'b' \\ &\quad -dbac' + dba'c' - dab'c' + da'b'c' \\ &\quad -cbad' + dcba'd' - cab'd' + ca'b'd' \\ &\quad +bac'd' - ba'c'd' + ab'c'd' - a'b'c'd'). \end{aligned} \tag{8}$$

In either case, the measure $D_4 = V_4/V$ is found through multiplication by the appropriate reciprocal frame vectors and summing. The grade 2 part of V/V_0 can then be gleaned from D_1 or D_2 above.

These examples in $\mathbb{R}^{4,1}$ show that alternative methods may be employed to retrieve the versor, involving data or computations of different grades. For exact data and computations, these lead to equivalent results. Numerically, or with noisy data, one method may be preferable, but this remains to be investigated.

4 EVEN VERSOR DETERMINATION IN *N*-D

The pattern above generalizes to the n-dimensional case.

1. We define the quantity:

$$D_k = \sum_{\text{basis k-blades } X} X^r\,(X\rfloor V)/V. \tag{9}$$

2. We observe that D_k is measurable from n vector correspondences by the recursion formula for a k-blade of the basis:

$$\begin{aligned} \big((X_{k-1} \wedge x_k)\rfloor V\big)/V &= \big(X_{k-1}\rfloor(x_k \rfloor V)\big)/V \\ &= \tfrac{1}{2}\big(X_{k-1}\rfloor(x_k V - V x_k)\big)/V, \end{aligned}$$

of which repeated application ultimately rewrites $(X\rfloor V)/V$ as a properly weighted sum of geometric products of k factors of the measured data x_i and x_i' $(i = 1,\cdots,n)$. Having n vector correspondences available then provides the correspondences of all $\binom{n}{k}$ independent basis k-blades.

3. On the other hand, D_k equals a weighted sum of the even grade parts of V:

$$D_k = \sum_{K=0,2,\cdots}^{n} \tbinom{K}{k} V_K/V \tag{10}$$

Proof: We adapt a result from [4] (his eq.(2.41)), replacing Hestenes' inner product by the contraction for the case $k \le K$:

$$\sum_{\text{basis k-blades } X} X^r\,(X\rfloor V_K) = \partial_{B_k}(B_k \rfloor V_K) = \tbinom{K}{k} V_K, \tag{11}$$

where B_k denotes the general k-vector.

4. Collect enough independent D_k to get a solvable system for all components of the even grades of V/V_0. This can always be done.

The simplest way to demonstrate this for a versor of highest possible grade $h = 2\lfloor n/2 \rfloor$ in n-D is to take the entries $D_2,\cdots,D_{h-2},D_h$, which depend linearly on the $V_2/V,\cdots,V_{h-2}/V,V_h/V$ through an upper triangular matrix which is obviously of full rank, since its leading entries are all equal to 1:

$$\begin{bmatrix} D_2 \\ D_4 \\ D_6 \\ D_8 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & 6 & 15 & 28 & \cdots \\ 0 & 1 & 15 & 70 & \cdots \\ 0 & 0 & 1 & 28 & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} V_2/V \\ V_4/V \\ V_6/V \\ V_8/V \\ \vdots \end{bmatrix}$$

Invert this relationship to solve for the V_i/V in terms of the D_k, giving

$$\begin{bmatrix} V_2/V \\ V_4/V \\ V_6/V \\ V_8/V \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & -6 & 75 & -1708 & \cdots \\ 0 & 1 & -15 & 350 & \cdots \\ 0 & 0 & 1 & -28 & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} D_2 \\ D_4 \\ D_6 \\ D_8 \\ \vdots \end{bmatrix}$$

(as is the pattern for inverses of upper triangular matrices, this inverse grows as k grows by adding rows and colums, but earlier entries do not change).

Since this simple procedure does not necessarily lead to the simplest expressions, a more complete overview of the relationship between the V_i/V and D_k is provided below.

5. If required, normalize the even versor to a rotor.

For any dimension, a closed form solution can now be constructed by the above procedure, or by alternative procedures based on different grades in step 4.

5 THE PATTERN FOR ODD AND EVEN BLADES

Using the even grades only in step 4 may lead to unnecessarily involved expressions. For instance, for Euclidean 3-space this procedure produces:

$$V/V_0 = (1 - V_2/V)^{-1} = (1 - D_2)^{-1} \propto 1 - \widetilde{D}_2,$$

as a 2-blade expression to 'compete' with the straightforward 1-blade expression $1 - \frac{1}{2}\widetilde{D}_1$ in eq.(3) based directly on the vector data. For the conformal model $\mathbb{R}^{4,1}$, the procedure gives:

$$\begin{aligned} V/V_0 &= (1 - V_2/V - V_4/V)^{-1} \\ &= (1 - D_2 + 5D_4)^{-1} \\ &\propto 1 - \widetilde{D}_2 + 5\widetilde{D}_4, \end{aligned}$$

to rival eq.(6). The latter is simpler to compute because it is based on lower grades (it avoids evaluating eq.(8)).

We give an extended table up to 8 dimensions for the coefficients in the identity eq.(10), to be used in alternative strategies:

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \\ D_7 \\ D_8 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \cdots \\ 0 & 2 & 4 & 6 & 8 & \cdots \\ 0 & 1 & 6 & 15 & 28 & \cdots \\ 0 & 0 & 4 & 20 & 56 & \cdots \\ 0 & 0 & 1 & 15 & 70 & \cdots \\ 0 & 0 & 0 & 6 & 56 & \cdots \\ 0 & 0 & 0 & 1 & 28 & \cdots \\ 0 & 0 & 0 & 0 & 8 & \cdots \\ 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} V_0/V \\ V_2/V \\ V_4/V \\ V_6/V \\ V_8/V \\ \vdots \end{bmatrix}$$

The row for D_0, representing the trivial identity $1 = V/V$, was added to show the symmetry of the table: its colums are even rows of Pascal's triangle, in accordance with eq.(10).

6 COMPLEXITY

In principle, we only need to determine $\binom{n}{2} = \frac{1}{2}n(n-1)$ parameters for an even versor in n-D.

In our algorithm, determining D_k from the data takes $k\,2^k$ operations: there are 2^k terms of k geometric products of n-vectors (though some use of recursive structure may be possible). If we only use the lowest grade D_k's to determine the versor, we need to compute all up to grade $\lfloor n/2 \rfloor$. Therefore the computational complexity of the algorithm is $\sum_{i=1}^{\lfloor n/2 \rfloor} k\,2^k$, i.e., $O(2^n)$.

In [6](pg.205), a numerical method is given to determine a versor from possibly noisy and redundant correspondence data in n-D. It does so by rephrasing the versor recovery in terms of a set of linear equations in the linear, 2^n-dimensional space of multivectors, and determining the null space of a $2^n \times 2^n$ matrix. Since this is in general an $O(2^{3n})$ procedure, it may be considerably more expensive than our $O(2^n)$ method (though perhaps the sparseness of the geometric product matrix can be exploited to reduce this). We emphasize that [6]'s method was designed for more exacting situations than the exact data that interest us in this paper.

In any case, compared to the $\frac{1}{2}n(n-1)$ parameters actually required to be extracted from n points in n-D, both methods seem rather expensive. One would expect a more efficient, perhaps even polynomial, algorithm to beat both these methods in the future.

7 ODD VERSORS

For odd versors, our method fails. For instance, for a versor consisting only of single term of grade 1, we obtain by eq.(11) $D_1 = V_1/V$, and the other D_i are zero. Therefore we can only determine $V_1/V = 1$, and this gives no information on the actual reflection vector involved. Essentially the same indeterminacy happens for arbitrary odd versors, and kills the method. (Actually, a similar indeterminacy occurred for the lowest grade term in even versors, since we could only determine V/V_0; but of course a mere scalar factor does not affect the versors determination in a crucial manner.)

However, we can still determine the odd versor from the data if we apply one step to convert the problem into that of determining an even versor. Taking any vector x that is changed to a different vector x', form $v = (x - x')$ as the versor that is capable of transforming x to x' (but mind that v should be non-null as well as nonzero, if it is not pick a different x). Then apply v to all original vectors x_i, giving intermediate vectors $v_i \equiv -v x_i / v$. Now find the even versor V that transforms all v_i into x'_i by the regular method. The total odd versor transforming the x_i into the x'_i is then Vv.

Incidentally, there is a simple check to determine whether an odd or an even versor is required to transform the n vectors x_i into the x'_i: just form the pseudoscalars $x_1 \wedge \cdots \wedge x_n$ and $x'_1 \wedge \cdots \wedge x'_n$. If their ratio is 1 look for an even versor, if it is -1 look for an odd versor.

8 AN ALTERNATIVE METHOD?

In [4](pg. 111), there is a method of 'finding the rotor S from the matrix of a rotation' (a rotor is a normalized even versor, for which $S\widetilde{S} = 1$). It is based on a rotor-induced linear mapping f being given as a matrix, i.e. a correspondence between the vectors x_i in a complete basis and their rotated versions $f(x_i)$. It is observed that the S-transformations of the x_i obey the relationship:

$$\sum_i x^i S x_i \widetilde{S} = \partial f = \sum_i x^i f(x_i) = \sum_{ij} f_{ij} x^i x^j. \qquad (12)$$

The right hand side involving f is fully computable from the correspondences $x_i \mapsto x'_i = f(x_i)$, and the left hand side involves S. Then [4] states that this equation for S can be solved in general by decomposing $\partial \wedge f$ into orthogonal blades. This general method would be completely different from our approach, and is actually not necessarily of closed form in n-D, due to the need to solve polynomial equations of degree n (which is in general not possible beyond $n = 4$, though Anthony Lasenby's recent usage of them up to $n = 10$ suggests that these particular equations may take a solvable form, but this remains unproven).

However, Hestenes and Sobczyk also state that 'for spaces of small dimension it is practical to solve eq.(12) directly for S', and this is closer to our goal. Unfortunately, they do not give a method, but merely an example for $n = 4$. They use the relationship $\dot{\partial} S_r \dot{x} = (-1)^r(n-2r)S_r$ to derive: $\sum_i x^i S x_i = \partial_x S x = 4(S_0 - S_4)$, so that $\partial f = 4(S_0 - S_4)\widetilde{S}$, and therefore $\partial f \widetilde{\partial f} = (4(S_0 - S_4))^2$. Therefore in this case of $n = 4$, if we can take a square root of an element consisting of (scalar + quadvector), we can determine the factor in front of $\widetilde{S}$ from the data on ∂f alone, and invert the equation as $\widetilde{S} = \pm\partial f/(\partial f \widetilde{\partial f})^{1/2}$ (with some exceptional special cases).

Generalizing this example to other dimensions is not as straightforward as [4]'s lack of detail seems to suggest. Already in the conformal model with $n = 5$, we find $\partial f = (5S_0 + S_2 - 3S_4)\widetilde{S}$, and $\partial f \widetilde{\partial f}$ now involves all grades 0, 2, 4, making the square root even harder to extract.

For $n = 3$, we get $\partial f = (3S_0 - S_2)\widetilde{S}$. Their sketched solution procedure now does not work, and the equation is most easily solved by observing the rather fortuitous fact that $\partial f = (3S_0 - S_2)(S_0 - S_2) = 3S_0^2 + S_2^2 - 4S_0S_2 = 4S_0(S_0 - S_2) - 1 = 4S_0\widetilde{S} - 1$, so that $\widetilde{S} \propto (1 + \sum x^i f(x_i))$, which is the 3D case of eq.(3) reported in [5].

The solvable cases appear to depend on rather clever manipulation or accidental factorization that does not generalize. It would seem that using eq.(12) one would have to resort to somehow setting up more equations on the higher order correspondences. Our method based on eq.(9) and eq.(10) gives a structured way of doing just that.

9 PRACTICAL CONSIDERATIONS

Under several circumstances occurring in practice, naive application of our method to the data may give the wrong result. We list some of those traps and discuss their resolution.

9.1 Exceptional Failure

There are situations in which a unique versor cannot be determined from the transformation data. This is for instance already the case in a 2-D rotation over π: taking an orthonormal basis $\mathbf{e}_1 \mapsto -\mathbf{e}_1$ and $\mathbf{e}_2 \mapsto -\mathbf{e}_2$, and the classical formula eq.(3) yields $V/V_0 = 0$.

I do not know what the conditions for failure are in a general geometric algebra. In the Euclidean model $\mathbb{R}^n$ (where versor are rotations) for even versors, they are precisely the failure conditions of eq.(3) which are a single rotation over π (of which the 2-D case above is the canonical example). In that case V/V_0 evaluates to zero, and therefore $V = 0$. To resolve this issue, one may pick an arbitrary normalized rotation plane $\mathbf{I}$, and form the versor $V = \exp(-\mathbf{I}\pi/2)$. In the conformal model $\mathbb{R}^{n+1,1}$ this is slightly generalized to translated rotations, and resolved by choosing the properly translated version of an arbitrary origin rotation plane $\mathbf{I}$.

9.2 Unknown Weights

Our method works by knowing where *vectors* have gone; in an application like the conformal model $\mathbb{R}^{m+1,1}$ of $\mathbb{E}^n$, where the vectors represent points, one cannot always measure that from the point data. For instance, in a uniform scaling around the origin by a factor α, it seems as though a point at location $\mathbf{p}$ goes to a point at location $\alpha\mathbf{p}$. However, this is not an orthogonal transformation of the conformal representation space $\mathbb{R}^{m+1,1}$, and can therefore not be described by a versor. As a consequence, reconstruction by our method fails. Using as input the naive $n_o \mapsto n_o$, $n_\infty \mapsto n_\infty$ and $\mathbf{e}_i \mapsto \alpha\mathbf{e}_i$, we obtain:

$$\begin{aligned} 4 - n + \sum x^{\prime} x^{r} &= \\ &= 4 - n + \sum \alpha\mathbf{e}_i\,\mathbf{e}_i + n_o(-n_\infty) + n_\infty(-n_o) \\ &= 6 - n + n\alpha \\ &\propto 1, \end{aligned}$$

which is clearly wrong.

When a uniform scaling versor is applied to conformal vectors, the conformal basis vectors actually change according to: $n_o \mapsto n_o/\alpha$, $n_\infty \mapsto \alpha n_\infty$ and $\mathbf{e}_i \mapsto \mathbf{e}_i$. Using this in reconstruction by eq.(3) (since we happen to know that the highest grade is 2) indeed produces the correct scaling versor:

$$\begin{aligned} 4 - n + \sum x^{\prime} x^{r} &= \\ &= 4 - n + \sum \mathbf{e}_i\,\mathbf{e}_i + n_o(-n_\infty)/\alpha + \alpha n_\infty(-n_o) \\ &= 4 + \cosh(\alpha) + \sinh(\alpha)\, n_o \wedge n_\infty \\ &= 2\cosh(\alpha/2)\left(\cosh(\alpha/2) + \sinh(\alpha/2)\, n_o \wedge n_\infty\right) \\ &\propto \exp(\alpha n_o \wedge n_\infty/2). \end{aligned}$$

As the vector transformation indicates, a unit-weight point $n_o + \mathbf{p} + \frac{1}{2}\mathbf{p}^2 n_\infty$ becomes the weighted point $(n_o + \alpha\mathbf{p} + \frac{1}{2}(\alpha\mathbf{p})^2 n_\infty)/\alpha$ at the location $\alpha\mathbf{p}$.

To apply the formulas of this paper to points in the conformal model, one therefore needs to observe not only point locations, but also their weights. Where this is an unrealistic assumption in practice, we recommend the alternative weightless method [2], especially designed for the conformal model. It employs a Vahlen matrix representation to reduce the conformal versor determination to purely Euclidean computations. Yet even that paper does not make the present contribution superfluous: it uses our method to make the Euclidean part of its computations extend to the m-D case required in $\mathbb{R}^{m+1,1}$.

9.3 Rigid Body Motions Covered

It should be noted that the method can be used to determine the weight-preserving transformations of the conformal model. Since the weight of a point represented by a vector p equals $-n_\infty \cdot p$, those are the transformations for which n_∞ is invariant. These are of course precisely the useful Euclidean transformations (though *not* the Euclidean similarities).

The determination of weight-preserving transformations can be extended to non-Euclidean geometries; we can choose our infinity i as an arbitrary vector in the conformal model rather than $i = n_\infty$, and define the weight of a point represented by a vector p as $-i \cdot p$. Then all i-preserving transformations can be determined by our method (since we need not observe their weights). By judicious choice of i, these then include the equivalents of translational and rotational motions in spherical geometry and in hyperbolic geometry (see e.g. [3], Chapter 16).

9.4 Nonexact Data

Our method does not use the versor nature of V explicitly. For non-exact or redundant data, it presumably degenerates gently, to produce a non-versor 'almost, but not quite, entirely unlike' the original V, and which nearly preserves grades. How to optimally correct such a multivector outcome to a versor is not yet known.

REFERENCES

[1] P. Anglès, *Construction de revêtements du group conform d'un espace vectorial muni d'une "métrique" de type (p,q).* Annales de l'Institut Henri Poincaré, Section A, Vol. XXXIII:33-51, 1980.

[2] C. Cibura, L. Dorst, *From Exact Correspondence Data to Conformal Transformations in Closed Form Using Vahlen Matrices*, Gravisma 2009.

[3] L. Dorst, D. Fontijne, S. Mann, *Geometric Algebra for Computer Science*, Morgan-Kauffmann, 2007.

[4] D. Hestenes, *Clifford Algebra to Geometric Calculus*, Reidel, Dordrecht, 1984.

[5] J. Lasenby, A.N. Lasenby, C.J.L. Doran, W.J. Fitzgerald, *New Geometric Methods for Computer Vision: an application to structure and motion estimation*, IJCV 36(3), pp.191-213, 1998.

[6] C. Perwass, *Geometric Algebra with Applications in Engineering*, Springer 2009.

Conformal Geometric Algebra by Extended Vahlen Matrices

Leo Dorst
University of Amsterdam, The Netherlands
L.Dorst@uva.nl

ABSTRACT

The classical Vahlen matrix representation of conformal transformations on $\mathbb{R}^n$ is directly related to the versor representation of conformal geometric algebra (CGA) using $\mathbb{R}^{n+1,1}$. This paper spells out the relationship, which enriches both fields with insights and techniques. We extend the Vahlen matrices to include the representation of blades in CGA, and then use a decomposition in terms of eigenlines to derive Chasles' theorem for representation of Euclidean rigid body motions. This naturally leads to the logarithm of a Vahlen matrix of such a motion. We also derive the table of commutation relationships between the basic even conformal transformations (translation, rotation, uniform scaling and transversion), in which the rather involved translation-transversion result may be new.

Keywords: conformal transformations, conformal geometric algebra, CGA, conformal model, Vahlen matrices, rigid body motions, Chasles' theorem, commutation relations

1 INTRODUCTION

The conformal transformations in $\mathbb{R}^n$ can be modeled conveniently as orthogonal transformations of $\mathbb{R}^{n+1,1}$ [1], and these in turn are representable as versors in geometric algebra (i.e. as geometric products of invertible vectors, see e.g [3]). Using anti-symmetric combinations of geometric products, an outer product can be introduced as the basis for the Grassmann algebra of $\mathbb{R}^{n+1,1}$. The outer products of not necessarily invertible vectors form *blades* representing subspaces of $\mathbb{R}^{n+1,1}$ that can be identified with circles, spheres, planes and tangents (and more) in the space $\mathbb{R}^n$. The versors act on them in a structure-preserving manner. Thus we get universal conformal operators, simplifying software. As a consequence the 'conformal model' (a.k.a. CGA, conformal geometric algebra) is beginning to be useful in computer science fields like graphics, vision and robotics, where the main interest is usually in Euclidean transformations rather than the general conformal mappings.

On the other hand, conformal transformations have been studied in mathematics using Vahlen matrices, as a homogeneous representation of Möbius transformations. The Vahlen matrices have been extended to allow coefficients from a Clifford algebra.

In this paper, we mix the two ideas, to lay the foundation for a fruitful interaction, expanding results in [7, 4]. On the one hand, we get a convenient mechanism for computations in the conformal model using the Vahlen matrices (though extended from transformations to the representation of geometric primitives), in which we never need more than the familiar Euclidean geometric algebra to do our conformal computations. Conversely, the geometrical semantics of the conformal model permits a study of, for instance, all conformal transformations that preserve a given line and hence informs the study of Möbius transformations, easily recovering Chasles' theorem. A full table of commutation relationshops between elementary even conformal motions is easily derived, notably a possibly new result on the commutation of translation and transversion.

2 THE CONFORMAL MODEL OF $\mathbb{R}^3$

The conformal model of $\mathbb{R}^3$ uses the geometric algebra of a Minkowski space of two more dimensions $\mathbb{R}^{4,1}$ to represent conformal transformations of $\mathbb{R}^3$. By the choice of correspondence between the two spaces and the metric of $\mathbb{R}^{4,1}$, conformal transformations of $\mathbb{R}^3$ are representable as orthogonal transformations of $\mathbb{R}^{4,1}$; and by the use of geometric algebra, those are representable as the product of (at most 5) invertible vectors. Such a product of invertible vector is called a *versor*, and it is applicable to all elements of the algebra by a sandwiching product. Geometrically, the vectors of $\mathbb{R}^{4,1}$ are interpretable as affine planes or general spheres (including points) in $\mathbb{R}^n$, and the versor representation is interpretable as multiple reflection in the corresponding elements. It is a computationally attractive implementation of the Cartan-Dieudonné insight that an orthogonal transformation of an m-dimensional metric space can be represented using at most m reflections.

The basic correspondence between Euclidean elements and their conformal representation is:

- The point at infinity is represented by a null vector $n_\infty \in \mathbb{R}^{4,1}$. 'Null vector' implies that $n_\infty^2 = 0$.
- A point X with weight α is represented as a null vector x such that $-n_\infty \cdot x = \alpha$.
- The squared Euclidean distance between two points X and Y is represented as

$$d_E^2(X,Y) = \left(\frac{x}{-n_\infty \cdot x}\right) \cdot \left(\frac{y}{-n_\infty \cdot y}\right).$$

As a consequence, Euclidean transformations are represented by orthogonal transformations in $R^{4,1}$ that preserve n_∞.

- A Euclidean geometric primitive (such as a plane or circle) is represented directly by the blade A if its points x satisfy $x \wedge A = 0$, and dually by the blade D if its points satisfy $x \rfloor D = 0$ (where $\rfloor$ is the contraction, an extension of the inner product of $\mathbb{R}^{4,1}$ to multivectors, see [3]).

The geometric primitives that can be represented in this manner may be found in the Table 1; they include *flats* (planes, lines, flat points); *rounds* (spheres, circles, point pairs); *tangents* (tangent planes, tangent vectors, weighted points); and *directions* (free vectors, free bivectors, free volumes).

2.1 Matrix Representation Principles

It is convenient to partition the Minkowski space $\mathbb{R}^{4,1}$ as $\mathbb{R}^{4,1} = \mathbb{R}^{1,1} \otimes \mathbb{R}^3$, with the $\mathbb{R}^3$ a 'copy' of the Euclidean space of directions we are modelling, and $\mathbb{R}^{1,1}$ being used for the metric aspects of location and size. We are going to model this structure as 2×2 matrices (encoding the $\mathbb{R}^{1,1}$ structure) with matrix entries taken from the geometric algebra of Euclidean $\mathbb{R}^3$ (encoding the $\mathbb{R}^3$ part). These include Vahlen matrices for the versors, but also null matrices for some of the geometric primitives. We use $\asymp$ to denote the correspondence, and bold font for the Euclidean elements.

The geometric product in $\mathbb{R}^{4,1}$ is then represented as the product of such matrices, as is usual for Vahlen matrices [7]. Representation of other products can be based on reductions to linear combinations of geometric products: the *outer product* of a vector x and a blade A is transferred by using $x \wedge A = \frac{1}{2}(xA + \widehat{A}x)$ (where the hat denotes grade involution); and the *contraction product* of a vector x and a blade A by using $x \rfloor A = \frac{1}{2}(xA - \widehat{A}x)$.

2.2 The core $\mathbb{R}^{1,1}$

A basis for $\mathbb{R}^{1,1}$ is e_+ and e_- such that $e_+^2 = 1$ and $e_-^2 = -1$, with $e_+ \cdot e_- = 0$. Picking a matrix representation of these elements as:

$$e_+ \asymp \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ and } e_- \asymp \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{1}$$

satisfies these relations (with the unit represented as the identity matrix). But putting a minus sign on either of these is also permitted (as are other variations).

Besides the identity $1 (= e_+^2 = -e_-^2)$, the other element in the Grassmann algebra basis for the geometric algebra of $\mathbb{R}^{1,1}$ is

$$E \equiv e_+ \wedge e_- \asymp \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
$$= \tfrac{1}{2}\left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right).$$

It is common in CGA treatments to prefer the use of a specific *null basis* for $\mathbb{R}^{1,1}$:

$$n_o = \tfrac{1}{2}(e_+ + e_-) \text{ and } n_\infty = e_- - e_+,$$

which satisfy $n_o \cdot n_\infty = -1$. Note that $n_o \wedge n_\infty = e_+ \wedge e_- = E$. Geometrically, n_∞ and n_o will represent the point at infinity, and the (arbitrarily chosen) origin.

Using the above matrices, this basis leads to the following matrix representation for $\mathbb{R}^{1,1}$:

$$\{1, n_o, n_\infty, E \equiv n_o \wedge n_\infty\} \asymp \tag{2}$$
$$\left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right\}.$$

We can then represent an arbitrary multivector of $\mathbb{R}^{1,1}$ with scalar coefficients as a matrix. The conversion back is:

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \asymp \tfrac{1}{2}(\alpha + \delta) + \tfrac{1}{2}(\alpha - \delta)E + \gamma n_o + \beta\,(-n_\infty/2). \tag{3}$$

We emphasize that this is defined for scalar coefficients.

2.3 Translated Points

In the work on Möbius transformations in Clifford algebras, the Vahlen matrix representing a translation is taken to be (see e.g. [7]):

$$T_{\mathbf{t}} \asymp \begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix}. \tag{4}$$

The theory usually deals only with the correspondences between the conformal transformations represented by these and other matrices. In searching for an embedding of the conformal model, we would like them to work as versors on the matrix representation of $\mathbb{R}^{1,1}$. The null vector n_o is taken as representing the point at the origin; therefore a point x at location is represented as:

$$x = T_{\mathbf{x}}\, n_o\, T_{\mathbf{x}}^{-1} \asymp \tag{5}$$
$$\begin{bmatrix} \mathbf{x} & -\mathbf{x}^2 \\ 1 & -\mathbf{x} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{x} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -\mathbf{x} \\ 0 & 1 \end{bmatrix}.$$

This matches [7, 4]. However, we see that naive application of the interpretation formula eq.(3) produces $x = n_o + \mathbf{x}E + \frac{1}{2}\mathbf{x}^2 n_\infty$, which does not match the usual embedding of [3] (though it does match Hestenes' conformal split, see Section 5). In fact, all matrices produced should be interpreted through the interpretation equation:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \asymp \tfrac{1}{2}(\mathbf{A} + \widehat{\mathbf{D}}) + \tfrac{1}{2}(\mathbf{A} - \widehat{\mathbf{D}})E + \mathbf{B}\,(-\tfrac{1}{2}n_\infty) + \widehat{\mathbf{C}}\, n_o. \tag{6}$$

We prove this below, in Section 5.

2.4 Involution, Reversion, Inverse, Dual

In the matrix representation, the sign change operators of *reversion* and *grade involution* are (see [7]):

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{\widehat{}} = \begin{bmatrix} \widehat{\mathbf{A}} & -\widehat{\mathbf{B}} \\ -\widehat{\mathbf{C}} & \widehat{\mathbf{D}} \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{\sim} = \begin{bmatrix} \bar{\mathbf{D}} & \bar{\mathbf{B}} \\ \bar{\mathbf{C}} & \bar{\mathbf{A}} \end{bmatrix}$$

where $\bar{\cdot}$ denotes Clifford conjugation, the grade involution of the reversion.

The *inverse* of an invertible versor (including blades) represented by a Vahlen matrix is largely as expected, though there are some signs to mind:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \frac{1}{\mathbf{A}\widetilde{\mathbf{D}} - \mathbf{B}\widetilde{\mathbf{C}}} \begin{bmatrix} \widetilde{\mathbf{D}} & -\widetilde{\mathbf{B}} \\ -\widetilde{\mathbf{C}} & \widetilde{\mathbf{A}} \end{bmatrix}$$

This is well defined, due to certain conditions that are placed on the Vahlen matrices. These conditions adapted to the specific case of a Vahlen matrix over the geometric algebra of the Euclidean space $\mathbb{R}^n$ are:

- $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are Euclidean versors (i.e. can be written as the products of invertible vectors from $\mathbb{R}^n$) or zero;
- $\widetilde{\mathbf{A}}\mathbf{B}$, $\widetilde{\mathbf{D}}\mathbf{C}$, $\mathbf{B}\widetilde{\mathbf{D}}$, $\mathbf{C}\widetilde{\mathbf{A}}$ are vectors from $\mathbb{R}^n$;
- the determinant $\mathbf{A}\widetilde{\mathbf{D}} - \mathbf{B}\widetilde{\mathbf{C}}$ is a (nonzero) scalar.

In this paper we also represent non-versor elements of the algebra my matrices, and drop the 'nonzero' demand from the final condition. We will call matrices satisfying the resulting conditions '*extended Vahlen matrices*'. Note that the matrices for n_o and n_∞ in eq.(1) are of this kind, as is x in eq.(5).

For *dualization* we use the pseudoscalar $n_o \wedge \mathbf{I}_n \wedge n_\infty = \widehat{\mathbf{I}}_n E$, following [3]. Constructing this dualization through multiplication by the element representing the inverse pseudoscalar $\widehat{\mathbf{I}}_n^{-1} E$, of which the representation is:

$$\widehat{\mathbf{I}}_n^{-1} E = \begin{bmatrix} \widehat{\mathbf{I}}_n^{-1} & 0 \\ 0 & -\mathbf{I}_n^{-1} \end{bmatrix}$$

we find for $X^* = X \widehat{\mathbf{I}}_n^{-1} E$:

$$X^* \asymp \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^* = \begin{bmatrix} (-1)^n \mathbf{A}^\star & -\mathbf{B}^\star \\ (-1)^n \mathbf{C}^\star & -\mathbf{D}^\star \end{bmatrix},$$

where $\cdot^\star$ denotes Euclidean dualization $\mathbf{X}^\star \equiv \mathbf{X}\mathbf{I}_n^{-1}$.

3 CGA ELEMENTS AS MATRICES

3.1 Versors as Vahlen Matrices

The difference between two (normalized) points in CGA is a vector (dually) representing their midplane. Using eq.(5) for two points at locations $\mathbf{n}/2$ and $-\mathbf{n}/2$, we find that this linear construction can be mimicked in the matrix representation, leading to the matrix of a plane through the origin with Euclidean normal vector $\mathbf{n}$ as:

$$\mathbf{n} \asymp \begin{bmatrix} \mathbf{n} & 0 \\ 0 & -\mathbf{n} \end{bmatrix}.$$

This is a Vahlen matrix; it can be used to represent the conformal transformation of reflecting in this plane. Applying a translation versor to $\mathbf{n}$ by 'sandwiching' leads to the general plane, and this can be mimicked in terms of the Vahlen matrices:

$$T_{\mathbf{p}}\,\mathbf{n}\,T_{\mathbf{p}}^{-1} = \mathbf{n} + (\mathbf{p}\cdot\mathbf{n})\,n_\infty \asymp$$
$$\begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{n} & 0 \\ 0 & -\mathbf{n} \end{bmatrix} \begin{bmatrix} 1 & -\mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & -2\mathbf{p}\cdot\mathbf{n} \\ 0 & -\mathbf{n} \end{bmatrix}$$

Combining the planes using the geometric product is represented as a product of matrices, and gives versors representing general rigid body motions. For instance, the matrix representation of the translation over $\mathbf{t}$ in eq.(4) is obtained as the reflection in two planes separated with unit normal vector $\mathbf{n} = \mathbf{t}/\|\mathbf{t}\|$ by $\|\mathbf{t}\|\mathbf{n}/2$. Taking the first reflection in a plane through the origin (for convenience), this gives:

$$T_{\mathbf{t}} \asymp \begin{bmatrix} \mathbf{n} & -\|\mathbf{t}\| \\ 0 & -\mathbf{n} \end{bmatrix} \begin{bmatrix} \mathbf{n} & 0 \\ 0 & -\mathbf{n} \end{bmatrix} = \begin{bmatrix} \mathbf{n}^2 & \|\mathbf{t}\|\mathbf{n} \\ 0 & \mathbf{n}^2 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix}$$

, consistent with eq.(4). By also permitting reflections in spheres we obtain the full set of conformal transformations. A sphere with center at location $\mathbf{c}$ and squared radius ρ^2 is (dually) represented in CGA as $c - \frac{1}{2}\rho^2 n_\infty$ (where c is the vector representing the point at location $\mathbf{c}$). This linear construction is easily transferred to extended Vahlen matrices, see Table 1.

It is customary to define some transformation primitives by arranging two reflectors in special ways. This leads to well-known Vahlen matrices for conformal transformations (naturally extending the classical Möbius transformations from the complex plane $\mathbb{C}$ to $\mathbb{R}^n$, see [7]). The resulting versors or matrices can be made as the product of the representations of the reflectors involved. In CGA, we commonly express the versors as the exponentials of 2-blades, so we give that notation as well.

- *Translation*: reflection in two parallel planes

$$T_{\mathbf{t}} \equiv e^{-\mathbf{t}n_\infty/2} = 1 - \mathbf{t}n_\infty/2 \asymp \begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix}.$$

- *Rotation*: reflection in two planes intersecting at the origin

$$\mathbf{R} = e^{-\mathbf{I}\phi/2} \asymp \begin{bmatrix} e^{-\mathbf{I}\phi/2} & 0 \\ 0 & e^{-\mathbf{I}\phi/2} \end{bmatrix} = \begin{bmatrix} \mathbf{R} & 0 \\ 0 & \mathbf{R} \end{bmatrix}.$$

- *Scaling*: reflection in two concentric spheres centered at the origin

$$S_{\exp(\gamma)} \equiv e^{\gamma E/2} \asymp \begin{bmatrix} e^{\gamma/2} & 0 \\ 0 & e^{-\gamma/2} \end{bmatrix}.$$

- *Transversion*: reflection in two equally large spheres tangent to each other at the origin

$$V_{\mathbf{v}} \equiv e^{n_o \mathbf{v}} = 1 + n_o \mathbf{v} \asymp \begin{bmatrix} 1 & 0 \\ \mathbf{v} & 1 \end{bmatrix}.$$

It is instructive to apply the transversion to a point:

$$\begin{aligned}
&\begin{bmatrix} 1 & 0 \\ \mathbf{v} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} & -\mathbf{x}^2 \\ 1 & -\mathbf{x} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\mathbf{v} & 1 \end{bmatrix} \\
&\quad = \begin{bmatrix} \mathbf{x} + \mathbf{x}^2\mathbf{v} & -\mathbf{x}^2 \\ 1 + \mathbf{x}^2\mathbf{v}^2 + 2\mathbf{v}\cdot\mathbf{x} & -\mathbf{x} - \mathbf{x}^2\mathbf{v} \end{bmatrix} \\
&\quad = \mathbf{x}^2(\mathbf{x}^{-1} + \mathbf{v})^2 \begin{bmatrix} (\mathbf{x}^{-1}+\mathbf{v})^{-1} & -(\mathbf{x}^{-1}+\mathbf{v})^{-2} \\ 1 & -(\mathbf{x}^{-1}+\mathbf{v})^{-1} \end{bmatrix},
\end{aligned}$$

giving both the location of the result at $(\mathbf{x}^{-1}+\mathbf{v})^{-1}$, and the scaling factor for the point weight.

The scaling versor $e^{\gamma E/2}$ applied to a point yields:

$$\begin{aligned}
&\begin{bmatrix} e^{\gamma/2} & 0 \\ 0 & e^{-\gamma/2} \end{bmatrix} \begin{bmatrix} \mathbf{x} & -\mathbf{x}^2 \\ 1 & -\mathbf{x} \end{bmatrix} \begin{bmatrix} e^{-\gamma/2} & 0 \\ 0 & e^{\gamma/2} \end{bmatrix} \\
&\quad = \begin{bmatrix} \mathbf{x} & -\mathbf{x}^2 e^{\gamma} \\ e^{-\gamma} & -\mathbf{x} \end{bmatrix} = e^{-\gamma} \begin{bmatrix} e^{\gamma}\mathbf{x} & -(e^{\gamma}\mathbf{x})^2 \\ 1 & -e^{\gamma}\mathbf{x} \end{bmatrix},
\end{aligned}$$

showing that the result is an $e^{-\gamma}$-weighted point at the location $e^{\gamma}\mathbf{x}$.

Using the transcription of the outer product and the contraction (inner) product to geometric products, we can easily find the representation of the geometrically significant blades from CGA in terms of extended Vahlen matrices (which do not necessarily satisfy the last Vahlen condition). Some specific examples and a complete symbolic list of these elements is given in Table 1.

3.2 Blades: Geometric Primitives

Blades are formed as outer products of vectors. The basic definition of the outer product of a vector x with a blade A is:

$$x \wedge A = \tfrac{1}{2}(xA + \widehat{A}x).$$

This construction can be applied iteratively to contruct blades of ever higher grade, starting from vectors. Since it is a linear combination of geometric products, the construction is easily transferred to the (extended) Vahlen matrices. As an example, we produce the flat point $x \wedge n_\infty$:

$$\begin{aligned}
x \wedge n_\infty &= \tfrac{1}{2}(x n_\infty - n_\infty x) \\
&\asymp \tfrac{1}{2}\left(\begin{bmatrix} \mathbf{x} & -\mathbf{x}^2 \\ 1 & -\mathbf{x} \end{bmatrix} \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} & -\mathbf{x}^2 \\ 1 & -\mathbf{x} \end{bmatrix} \right) \\
&= \tfrac{1}{2}\left(\begin{bmatrix} 0 & -2\mathbf{x} \\ 0 & -2 \end{bmatrix} - \begin{bmatrix} -2 & 2\mathbf{x} \\ 0 & 0 \end{bmatrix} \right) = \begin{bmatrix} 1 & -2\mathbf{x} \\ 0 & -1 \end{bmatrix}.
\end{aligned}$$

It is often more convenient to use the covariant structure of CGA, and construct a blade at the origin, moving that to a general location using the translation versor. That is why Table 1 lists both the origin form, and the general form, of the catalogue of all blades.

3.3 Derivatives

The vector derivative is a vector operator; it can also be represented as a Vahlen-like matrix, with differentiation operators as elements. Denoting differentiation with repsect to the coefficient x_o of n_o by $\partial_o \equiv \frac{\partial}{\partial x_o}$ and to the coefficient x_∞ of n_∞ by $\partial_\infty \equiv \frac{\partial}{\partial x_\infty}$ (which are both scalar operators), and differentiation to the Euclidean components by ∇ (a vector operator), we obtain:

$$\partial = -n_\infty \partial_o - n_o \partial_\infty + \nabla \asymp \begin{bmatrix} \nabla & 2\partial_o \\ -\partial_\infty & -\nabla \end{bmatrix}.$$

With the representation of a vector field $x = x_o n_o + \mathbf{x} + x_\infty n_\infty$ of $\mathbb{R}^{n+1,1}$ by the corresponding matrix, we can obtain results like:

$$\begin{aligned}
&\partial_x x = n + 2 \asymp \\
&\quad \begin{bmatrix} \nabla & 2\partial_o \\ -\partial_\infty & -\nabla \end{bmatrix} \begin{bmatrix} \mathbf{x} & -2x_\infty \\ x_o & -\mathbf{x} \end{bmatrix} \\
&\quad = \begin{bmatrix} \nabla\mathbf{x} + 2\partial_o x_o & 0 \\ 0 & \nabla\mathbf{x} + 2\partial_\infty x_\infty \end{bmatrix} = \begin{bmatrix} n+2 & 0 \\ 0 & n+2 \end{bmatrix},
\end{aligned}$$

Multivector derivatives may be developed along similar patterns.

4 CGA MATRIX COMPUTATIONS

We present some examples of computations in the conformal model using the matrix representation.

4.1 Euclidean transformations

Among the conformal transformations, the Euclidean rigid body motions are of particular interest. These are the conformal versors that preserve the point at infinity. We relax this condition slightly, permitting preservation modulo scaling. The general matrix form can then be derived from this demand:

$$\begin{aligned}
\begin{bmatrix} 0 & -2\alpha \\ 0 & 0 \end{bmatrix} &= \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} \\
&= \frac{1}{\mathbf{A}\widetilde{\mathbf{D}} - \mathbf{B}\widetilde{\mathbf{C}}} \begin{bmatrix} -2\mathbf{A}\widehat{\widetilde{\mathbf{C}}} & -2\mathbf{A}\widehat{\widetilde{\mathbf{A}}} \\ -2\mathbf{C}\widehat{\widetilde{\mathbf{C}}} & -2\mathbf{C}\widehat{\widetilde{\mathbf{A}}} \end{bmatrix}.
\end{aligned}$$

We find that $\mathbf{C} = 0$, and the only remaining entry gives $\alpha = \widetilde{\mathbf{D}}^{-1}\widehat{\widetilde{\mathbf{A}}}$. We apply the Vahlen condition that $\mathbf{B}\widetilde{\mathbf{A}}$ is a vector, say $\mathbf{B}\widetilde{\mathbf{A}} = \mathbf{t}(\widehat{\mathbf{A}}\widetilde{\mathbf{A}})$ (with the scalar proportionality factor chosen to reduce signs later on). Then introducing the normalized versor $\mathbf{U}$ through $\mathbf{A} = \sqrt{\pm\alpha}\,\mathbf{U}$, we find that the general versor keeping n_∞ invariant modulo scale is of the form:

$$\begin{bmatrix} \sqrt{\pm\alpha}\,\mathbf{U} & \sqrt{\pm\alpha}\,\mathbf{t}\widehat{\mathbf{U}} \\ 0 & \widehat{\mathbf{U}}/\sqrt{\pm\alpha} \end{bmatrix} = \begin{bmatrix} \sqrt{\pm\alpha} & 0 \\ 0 & 1/\sqrt{\pm\alpha} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U} & 0 \\ 0 & \widehat{\mathbf{U}} \end{bmatrix}.$$

element class	origin blade or versor	matrix at origin	CGA blade or versor at p	matrix at location $\mathbf{p}$
weighted point	αn_o	$\begin{bmatrix} 0 & 0 \\ \alpha & 0 \end{bmatrix}$	αp	$\alpha \begin{bmatrix} \mathbf{p} & -\mathbf{p}^2 \\ 1 & -\mathbf{p} \end{bmatrix}$
dual plane	$\mathbf{n}$	$\begin{bmatrix} \mathbf{n} & 0 \\ 0 & -\mathbf{n} \end{bmatrix}$	$p \rfloor (\mathbf{n} \wedge n_\infty)$	$\begin{bmatrix} \mathbf{n} & -2\mathbf{p}\cdot\mathbf{n} \\ 0 & -\mathbf{n} \end{bmatrix}$
dual sphere	$n_o - \frac{1}{2}\rho^2 n_\infty$	$\begin{bmatrix} 0 & \rho^2 \\ 1 & 0 \end{bmatrix}$	$p - \frac{1}{2}\rho^2 n_\infty$	$\begin{bmatrix} \mathbf{p} & -\mathbf{p}^2+\rho^2 \\ 1 & \mathbf{p} \end{bmatrix}$
flat point	$n_o \wedge n_\infty$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$p \wedge n_\infty$	$\begin{bmatrix} 1 & -2\mathbf{p} \\ 0 & -1 \end{bmatrix}$
line	$n_o \wedge \mathbf{u} \wedge n_\infty$	$\begin{bmatrix} -\mathbf{u} & 0 \\ 0 & -\mathbf{u} \end{bmatrix}$	$p \wedge \mathbf{u} \wedge n_\infty$	$\begin{bmatrix} -\mathbf{u} & -2\mathbf{p}\wedge\mathbf{u} \\ 0 & -\mathbf{u} \end{bmatrix}$
$\mathbf{A}$-flat	$n_o \wedge \mathbf{A} \wedge n_\infty$	$\begin{bmatrix} \widehat{\mathbf{A}} & 0 \\ 0 & -\mathbf{A} \end{bmatrix}$	$p \wedge \mathbf{A} \wedge n_\infty$	$\begin{bmatrix} \widehat{\mathbf{A}} & -2\mathbf{p}\wedge\mathbf{A} \\ 0 & -\mathbf{A} \end{bmatrix}$
dual $\mathbf{A}$-flat	$\mathbf{D} = \widehat{\mathbf{A}}^\star$	$\begin{bmatrix} \mathbf{D} & 0 \\ 0 & \widehat{\mathbf{D}} \end{bmatrix}$	$p \rfloor (\mathbf{D} \wedge n_\infty)$	$\begin{bmatrix} \mathbf{D} & 2\mathbf{p}\rfloor\widehat{\mathbf{D}} \\ 0 & \widehat{\mathbf{D}} \end{bmatrix}$
round (carrier $\mathbf{A}$)	$(n_o + \frac{1}{2}\rho^2 n_\infty)\,\mathbf{A}$	$\begin{bmatrix} 0 & -\rho^2\widehat{\mathbf{A}} \\ \mathbf{A} & 0 \end{bmatrix}$	$(p + \frac{1}{2}\rho^2 n_\infty)(p\rfloor(\mathbf{A}n_\infty))$	$\begin{bmatrix} \mathbf{pA} & -\mathbf{pAp} - \rho^2\widehat{\mathbf{A}} \\ 1 & -\mathbf{Ap} \end{bmatrix}$
dual round (carrier $\mathbf{D}^{-\star}$)	$(n_o - \frac{1}{2}\rho^2 n_\infty)\,\mathbf{D}$	$\begin{bmatrix} 0 & \rho^2\widehat{\mathbf{D}} \\ \mathbf{D} & 0 \end{bmatrix}$	$(p - \frac{1}{2}\rho^2 n_\infty)(p\rfloor(\mathbf{D}n_\infty))$	$\begin{bmatrix} \mathbf{pD} & -\mathbf{pDp} + \rho^2\widehat{\mathbf{D}} \\ 1 & -\mathbf{Dp} \end{bmatrix}$
tangent null blade	$n_o \wedge \mathbf{A}$	$\begin{bmatrix} 0 & 0 \\ \mathbf{A} & 0 \end{bmatrix}$	$-p\rfloor(p \wedge \mathbf{A} \wedge n_\infty)$	$\begin{bmatrix} \mathbf{pA} & -\mathbf{pAp} \\ \mathbf{A} & -\mathbf{Ap} \end{bmatrix}$
direction null blade	$\mathbf{A} \wedge n_\infty$	$\begin{bmatrix} 0 & -2\mathbf{A} \\ 0 & 0 \end{bmatrix}$	$\mathbf{A} \wedge n_\infty$	$\begin{bmatrix} 0 & -2\mathbf{A} \\ 0 & 0 \end{bmatrix}$
translation versor $T_\mathbf{t}$	$e^{-\mathbf{t}n_\infty/2}$	$\begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix}$	$e^{-\mathbf{t}n_\infty/2}$	$\begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix}$
rotation versor $R_{\mathbf{I}\phi}$	$\mathbf{R} = e^{-\mathbf{I}\phi/2}$	$\begin{bmatrix} \mathbf{R} & 0 \\ 0 & \mathbf{R} \end{bmatrix}$	$-p\rfloor(\mathbf{R} \wedge n_\infty)$	$\begin{bmatrix} \mathbf{R} & 2\mathbf{p}\rfloor\mathbf{R} \\ 0 & \mathbf{R} \end{bmatrix}$
scaling versor S_{e^γ}	$e^{\gamma n_o \wedge n_\infty/2}$	$\begin{bmatrix} e^{\gamma/2} & 0 \\ 0 & e^{-\gamma/2} \end{bmatrix}$	$e^{\gamma p \wedge n_\infty/2}$	$\begin{bmatrix} e^{\gamma/2} & -2\mathbf{p}\sinh(\gamma/2) \\ 0 & e^{-\gamma/2} \end{bmatrix}$
transversion versor $V_\mathbf{v}$	$e^{n_o\mathbf{v}}$	$\begin{bmatrix} 1 & 0 \\ \mathbf{v} & 1 \end{bmatrix}$	$e^{-p\rfloor(p\wedge\mathbf{v}\wedge n_\infty)}$	$\begin{bmatrix} 1+\mathbf{pv} & 0 \\ \mathbf{v} & 1 \end{bmatrix}$
derivative	$\partial_\mathbf{x}$	$\begin{bmatrix} \nabla & 2\partial_o \\ -\partial_\infty & -\nabla \end{bmatrix}$		

Table 1: A dictionary of the matrix equivalences of CGA blades and versors. Common specific examples at the top; general forms of all classes lower down. Bold elements are purely Euclidean.

The simplest such versor is obtained by taking $\mathbf{U} = 1$, providing a scaled translation. Taking $\mathbf{U} = \mathbf{n}$, a unit vector, we obtain a scaled translated reflection in a plane through the origin. Taking $\mathbf{U} = \mathbf{R}$, a Euclidean rotor, we find a scaled translated rotation in a plane through the origin (at least, in 3D, where an origin rotation defines a unique plane).

When we demand exact preservation of the point at infinity by setting $\alpha = 1$, we obtain Euclidean transformations and their well-known decomposition into a rotation followed by a translation. Permitting non-unit α augments this to Euclidean similarities.

4.2 Eigenlines

With the matrix representation of a general line as indicated in the Table 1, we can ask for the oriented eigenlines of the Euclidean transformations, solving for $\mathbf{p}$ and $\mathbf{u}$ in

$$\begin{aligned}\begin{bmatrix} \mathbf{u} & 2\mathbf{p}\wedge\mathbf{u} \\ 0 & \mathbf{u} \end{bmatrix} &= \\ &= \begin{bmatrix} \mathbf{R} & \mathbf{tR} \\ 0 & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{u} & 2\mathbf{p}\wedge\mathbf{u} \\ 0 & \mathbf{u} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{R}} & -\widetilde{\mathbf{R}}\mathbf{t} \\ 0 & \widetilde{\mathbf{R}} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{Ru}\widetilde{\mathbf{R}} & 2\mathbf{t}\wedge(\mathbf{Ru}\widetilde{\mathbf{R}}) + 2\mathbf{R}(\mathbf{p}\wedge\mathbf{u})\widetilde{\mathbf{R}}) \\ 0 & \mathbf{Ru}\widetilde{\mathbf{R}} \end{bmatrix}\end{aligned}$$

It follows from the diagonal entries that $\mathbf{Ru}\widetilde{\mathbf{R}} = \mathbf{u}$ so that $\mathbf{u}\rfloor\mathbf{R} = 0$. The geometric meaning of this in 3D is that $\mathbf{u}$ should be perpendicular to the rotation plane, which determines $\mathbf{u}$ (modulo scaling) (and in n-D this orthogonality demand specifies a $(n-2)$-D family of possible $\mathbf{u}$). If $\mathbf{R}$ has no grade-2 part, any $\mathbf{u}$ will do; since such an $\mathbf{R}$ is either the identity or a rotation over 2π, this makes sense.) Using $\mathbf{Ru}\widetilde{\mathbf{R}} = \mathbf{u}$, the off-diagonal gives:

$$0 = (\mathbf{p} - \mathbf{Rp}\widetilde{\mathbf{R}} - \mathbf{t}) \wedge \mathbf{u} = \Big(2(\mathbf{p}\rfloor\mathbf{R})\widetilde{\mathbf{R}} - (\mathbf{t}\wedge\mathbf{u})/\mathbf{u}\Big)\,\mathbf{u},$$

where we used that $\mathbf{u}\rfloor\big((\mathbf{p}\rfloor\mathbf{R})\widetilde{\mathbf{R}}\big) = 0$, which is easily verified. Therefore the necessary and sufficient condition for $\mathbf{p}$ to be the support vector of an invariant line in direction $\mathbf{u}$ is:

$$2(\mathbf{p}\rfloor\mathbf{R})\widetilde{\mathbf{R}} = (\mathbf{t}\wedge\mathbf{u})/\mathbf{u}. \qquad (7)$$

The lhs is a vector in the $\mathbf{R}$-plane (which moreover depends only on the $\mathbf{R}$-plane component of $\mathbf{p}$), so this equation can only be solved if the rhs is a vector in the $\mathbf{R}$-plane as well, or zero. Zero obtains for nonzero $\mathbf{t}$ iff when $\mathbf{t}$ is parallel to $\mathbf{u}$, so in the specific situation that $\mathbf{t}\rfloor\mathbf{R} = 0$, which is not a general rigid body motion. The non-zero solution for $\mathbf{p}\rfloor\mathbf{R}$ is only guaranteed for general $\mathbf{t}$ and $\mathbf{R}$ if we are dealing with a 3-dimensional Euclidean space (since then $\mathbf{t}\rfloor\mathbf{R} \neq 0$ in general, but already in 4D this is not the case). From now on we continue the computation in 3D, since that is our main interest anyway.

Using eq.(7), we can rewrite the original 3D rigid body motion in terms of its axis as:

$$\begin{aligned}\begin{bmatrix} \mathbf{R} & \mathbf{tR} \\ 0 & \mathbf{R} \end{bmatrix} &= \begin{bmatrix} \mathbf{R} & ((\mathbf{t}\cdot\mathbf{u})/\mathbf{u} + (\mathbf{t}\wedge\mathbf{u})/\mathbf{u})\mathbf{R} \\ 0 & \mathbf{R} \end{bmatrix} \\ &= \begin{bmatrix} 1 & (\mathbf{t}\cdot\mathbf{u})/\mathbf{u} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & 2\mathbf{p}\rfloor\mathbf{R} \\ 0 & \mathbf{R} \end{bmatrix} \\ &= \begin{bmatrix} 1 & \tau\mathbf{u} \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 1 & \mathbf{p} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & 0 \\ 0 & \mathbf{R} \end{bmatrix} \begin{bmatrix} 1 & -\mathbf{p} \\ 0 & 1 \end{bmatrix} \right),\end{aligned}$$

with $\tau = \mathbf{t}\cdot\mathbf{u}^{-1}$.

4.3 Chasles' Theorem

Comparing the above to Table 1 shows that a 3D rigid body motion can be represented as a translated rotation around an axis $p \wedge \mathbf{u} \wedge n_\infty$, followed by a translation in the direction of that axis (or vice versa, because the two transformations commute due to $\tau\mathbf{u}\rfloor(\mathbf{p}\rfloor\mathbf{R}) = -\tau\mathbf{p}\rfloor(\mathbf{u}\rfloor\mathbf{R}) = 0$). That is *Chasles' theorem.*

We can be more specific in computing the parameters $\mathbf{p}$ and $\mathbf{u}$ from the original $\mathbf{t}$ and $\mathbf{R}$. Defining $\mathbf{I} \equiv \langle\mathbf{R}\rangle_2/\|\langle\mathbf{R}\rangle_2\|$ to characterize the rotation plane, we obtain $\mathbf{u} = \mathbf{I}^\star$, the Euclidean dual of $\mathbf{I}$ (or a scalar multiple). Then define $\mathbf{p}_\| = (\mathbf{p}\wedge\mathbf{u})/\mathbf{u} = (\mathbf{p}\rfloor\mathbf{I})/\mathbf{I}$ and find $2(\mathbf{p}\rfloor\mathbf{R})\widetilde{\mathbf{R}} = (1 - \mathbf{R}^2)\mathbf{p}_\| = (\mathbf{t}\rfloor\mathbf{I})/\mathbf{I}$ so that $\mathbf{p} = (1-\mathbf{R}^2)^{-1}(\mathbf{t}\rfloor\mathbf{I})/\mathbf{I} + \lambda\mathbf{I}^\star$. It is customary to choose $\lambda = 0$, obtaing the orthogonal support vector $\mathbf{p} = (1-\mathbf{R}^2)^{-1}(\mathbf{t}\rfloor\mathbf{I})/\mathbf{I}$.

4.4 Logarithm of a 3D Rigid Body Motion

Using the above results, we can derive the logarithm of a rigid body motion. In terms of Vahlen matrices, this logarithm is a matrix of which the matrix exponential would equal a given rigid body motion matrix.

To treat this, we first compute the logarithm of a pure rotation. Focussing on the principal value (denoted Log) in 3D, this is based on retrieving $-\mathbf{I}\phi/2$ from the given rotor $\mathbf{R} = \exp(-\mathbf{I}\phi/2) = \cos(\phi/2) - \mathbf{I}\sin(\phi/2)$. This takes some straightforward splitting into known scalar functions and the 2-blade part:

$$\mathrm{Log}(\mathbf{R}) = \mathrm{atan}(\frac{\|\langle\mathbf{R}\rangle_2\|}{\langle\mathbf{R}\rangle_0})\,\frac{\langle\mathbf{R}\rangle_2}{\|\langle\mathbf{R}\rangle_2\|}.$$

This formula is not well-defined when $\mathbf{R}$ is scalar. We augment it by demanding that $\mathrm{Log}(\mathbf{R})$ equals 0 when $\mathbf{R} = 1$, and is not defined when $\mathbf{R} = -1$ (alternatively, one could select an arbitrary 2-blade $\mathbf{I}$ and write it as $\mathbf{R} = \exp(-\mathbf{I}\pi/2)$).

For general 3D rigid body motions, we use Chasles' theorem to split the total problem into manageable subparts:

$$\begin{aligned}\mathrm{Log}(T_{\mathbf{t}}\mathbf{R}) &= \mathrm{Log}\Big(T_{\mathbf{t}_\|}(T_{\mathbf{t}_\perp}\mathbf{R}T_{-\mathbf{t}_\perp})\Big) \\ &= \mathrm{Log}(T_{\mathbf{t}_\|}) + T_{\mathbf{t}_\perp}\,\mathrm{Log}(\mathbf{R})\,T_{-\mathbf{t}_\perp},\end{aligned}$$

where geometrically $\mathbf{t}_{\|}$ and $\mathbf{t}_{\perp}$ are the components of $\mathbf{t}$ parallel and perpendicular to the rotation plane, respectively. Algebraically, they satisfy $T_{\mathbf{t}_{\perp}} \operatorname{Log}(\mathbf{R}) = \operatorname{Log}(\mathbf{R})\, T_{\mathbf{t}_{\perp}}$ and $T_{\mathbf{t}_{\|}} \operatorname{Log}(\mathbf{R}) = -\operatorname{Log}(\mathbf{R})\, T_{\mathbf{t}_{\|}}$. Substituting the results for $\mathbf{t}_{\|}$ and $\mathbf{t}_{\perp}$ from our treatment of the Chasles theorem then gives:

$$\operatorname{Log}(\begin{bmatrix} \mathbf{R} & \mathbf{tR} \\ 0 & \mathbf{R} \end{bmatrix}) =$$

$$\begin{bmatrix} \operatorname{Log}(\mathbf{R}) & (\mathbf{t} \wedge \operatorname{Log}(\mathbf{R}))/\operatorname{Log}(\mathbf{R}) + (1-\mathbf{R}^2)^{-1}(\mathbf{t} \rfloor \operatorname{Log}(\mathbf{R})) \\ 0 & \operatorname{Log}(\mathbf{R}) \end{bmatrix}$$

We have thus found the logarithm of a Vahlen matrix for a rigid body motion by appealing to the CGA method for computing it from [3].

4.5 Commutation Rules

Using the Vahlen matrices, commutation rules between versors of various types are easily established. For instance, the familiar $T_{\mathbf{t}}R = RT_{\widetilde{R}\mathbf{t}R}$ is immediate:

$$\begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} = \begin{bmatrix} R & \mathbf{t}R \\ 0 & R \end{bmatrix} = \begin{bmatrix} R & R\widetilde{R}\mathbf{t}R \\ 0 & R \end{bmatrix}$$
$$= \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} 1 & \widetilde{R}\mathbf{t}R \\ 0 & 1 \end{bmatrix}.$$

Most other commutation rules are of this simple type where the parametrization of one versor is affected by the other versor in such a swapping rule, easily obtained from matrix factorization. They are summarized in Table 2.

The only swapping rule that is truly involved is the commutation rule between transversion and translation, which also drags in a rotation scaling. It can be based on the matrix identity:

$$\begin{bmatrix} 1 & 0 \\ \mathbf{v} & 1 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} (1+\mathbf{tv})^{-1} & 0 \\ 0 & 1+\mathbf{vt} \end{bmatrix} *$$
$$\begin{bmatrix} 1 & \mathbf{t}(1+\mathbf{vt}) \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ (1+\mathbf{vt})^{-1}\mathbf{v} & 1 \end{bmatrix}$$

(where $*$ denotes matrix multiplication) which may be verified by simply expanding both sides. The last factor is a transversion by the vector $(1+\mathbf{vt})^{-1}\mathbf{v} = 1/(\mathbf{t}+1/\mathbf{v})$ (which is the transversion of the translation vector). The middle factor is a translation by the vector $\mathbf{t}+\mathbf{tvt}$. The first factor is a rotation/scaling matrix at the origin, the rotational part is the rotor $(1+\mathbf{vt})/\|1+\mathbf{vt}\|$, and the scaling factor is $1/\|1+\mathbf{vt}\|^2$. For the other order (first transversion, then translation), we have:

$$\begin{bmatrix} 1 & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \mathbf{v} & 1 \end{bmatrix} = \begin{bmatrix} 1+\mathbf{tv} & 0 \\ 0 & (1+\mathbf{vt})^{-1} \end{bmatrix} *$$
$$\begin{bmatrix} 1 & 0 \\ \mathbf{v}(1+\mathbf{tv}) & 1 \end{bmatrix} * \begin{bmatrix} 1 & (1+\mathbf{tv})^{-1}\mathbf{t} \\ 0 & 1 \end{bmatrix}$$

. I had been trying to derive this particular commutation relationship within the usual CGA representations in vain; switching to the Vahlen representation made it very easy to find and prove. Although we can now clearly redo the proof in the usual CGA, this is the first example of a derivation that was helped by the Vahlen representation.

4.6 Determining a Conformal Transformation from Data

In [2], the Vahlen matrix representation is used to give a closed form solution for the conformal transformation transforming a localized frame and a point to a desired target, in n-D. It is the only known method that can deal with the lack of weight information in input data, and still recover scaling and transversion parts of the conformal transformation they have undergone. Though it, too, can now be rephrased in the usual CGA formulation, working with purely Euclidean elements somehow made it more easily obtainable.

5 THE CONFORMAL SPLIT

In his original 'Design' paper [4], Hestenes proposes a conformal split in which the elements of the Euclidean space are treated as *trivectors*: to embed a vector $\mathbf{x}$ he uses $\mathbf{X} \equiv \mathbf{x}E$, etc.

We note that the geometric product of vectors is preserved:

$$\mathbf{XY} = (\mathbf{x}E)\,(\mathbf{y}E) = \mathbf{xy},$$

and this extends to more general multivectors. Using this, there is a rather natural definition of the outer product:

$$\mathbf{X} \wedge \mathbf{Y} = \tfrac{1}{2}(\mathbf{XY} - \mathbf{YX}) = \mathbf{x} \wedge \mathbf{y},$$

which then replaces the troublesome $\mathbf{X} \wedge \mathbf{Y} = (\mathbf{x} \wedge E) \wedge (\mathbf{y} \wedge E) = 0$. The inner product is consistent anyway, but follows the same pattern:

$$\mathbf{X} \cdot \mathbf{Y} = \tfrac{1}{2}(\mathbf{XY} + \mathbf{YX}) = \mathbf{x} \cdot \mathbf{y}.$$

With this understanding, the geometric algebra of the trivectors is completely isomorhpic to that of the original Euclidean vectors. Even elements get a replacement of non-caps by caps, and odd elements get an additional E (left or right does not matter since the Euclidean elements commute with E).

So why bother? The point is that the commutation relationships between these Euclidean trivectors and the elements of $\mathbb{R}^{1,1}$ are much simpler than that of Euclidean vectors:

$$\mathbf{X}n_o = \mathbf{x}E\,n_o = -\mathbf{x}n_o\,E = n_o\,\mathbf{x}E = n_o\,\mathbf{X}$$

and similarly for n_∞. Therefore these trivectors commute with the core $\mathbb{R}^{1,1}$. As a consequence, we have no awkward potential sign changes in applying the interpretation equation eq.(3) to non-scalars.

	rotation R	scaling S	translation T	transversion V
R	$R_{\mathbf{I}\phi}R = RR_{\widetilde{R}\mathbf{I}\phi R}$	$RS_\sigma = S_\sigma R$	$RT_{\mathbf{t}} = T_{R\mathbf{t}\widetilde{R}}R$	$RV_{\mathbf{v}} \;=\; V_{R\mathbf{v}\widetilde{R}}R$
S	$S_\sigma R = RS_\sigma$	$S_\sigma S_\tau = S_\tau S_\sigma$	$S_\sigma T_{\mathbf{t}} = T_{\sigma\mathbf{t}}S_\sigma$	$S_\sigma V_{\mathbf{v}} \;=\; V_{\mathbf{v}/\sigma}S_\sigma$
T	$T_{\mathbf{t}}R = RT_{\widetilde{R}\mathbf{t}R}$	$T_{\mathbf{t}}S_\sigma = S_\sigma T_{\mathbf{t}/\sigma}$	$T_{\mathbf{t}}T_{\mathbf{s}} = T_{\mathbf{s}}T_{\mathbf{t}}$	$T_{\mathbf{t}}V_{\mathbf{v}} \;=\; A^{-1}V_{\mathbf{v}(1+\mathbf{tv})}T_{(1+\mathbf{tv})^{-1}\mathbf{t}}$
V	$V_{\mathbf{v}}R = RV_{\widetilde{R}\mathbf{v}R}$	$V_{\mathbf{v}}S_\sigma = S_\sigma V_{\sigma\mathbf{v}}$	$V_{\mathbf{v}}T_{\mathbf{t}} = AT_{\mathbf{t}(1+\mathbf{vt})}V_{(1+\mathbf{vt})^{-1}\mathbf{v}}$	$V_{\mathbf{v}}V_{\mathbf{w}} \;=\; V_{\mathbf{w}}V_{\mathbf{v}}$

Table 2: Commutation of even versors in the conformal model. In the entries for VT and TV, A is the rotation/scaling $A \asymp \text{diag}[(1+\mathbf{tv})^{-1}, 1+\mathbf{vt}]$, see text.

The matrix-to-multivector interpretation

$$\left[\!\!\left[\begin{array}{cc}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{array}\right]\!\!\right] \asymp \tfrac{1}{2}(\mathbf{A}+\mathbf{D}) + \tfrac{1}{2}(\mathbf{A}-\mathbf{D})E + \mathbf{C}n_o + \mathbf{B}(-n_\infty/2)$$

is valid for all multivectors, as long as we replace the occurrence of the Euclidean vectors in the multivector by their trivectors (and propagate this through the algebra).

This insight can in fact be worked into a proof for the Euclidean vector embedding eq.(6) above. We note that due to the embedding of the conformal versors, either $\{\mathbf{A}$ and $\mathbf{D}\}$ or $\{\mathbf{B}$ and $\mathbf{C}\}$ are simultaneously odd, with the other pair then being even. In the first case, we write $\mathbf{A} = AE, \mathbf{B} = B, \mathbf{C} = C, \mathbf{D} = DE$ and find

$$\begin{aligned}
&\tfrac{1}{2}(\mathbf{A}+\mathbf{D}) + \tfrac{1}{2}(\mathbf{A}-\mathbf{D})E + \mathbf{C}n_o + \mathbf{B}(-n_\infty/2) = \\
&\quad = \tfrac{1}{2}(A+D)E + \tfrac{1}{2}(A-D) + Cn_o + B(-n_\infty/2)\\
&\quad = \tfrac{1}{2}(A+\widehat{D}) + \tfrac{1}{2}(A-\widehat{D})E + \widehat{C}n_o + B(-n_\infty/2).
\end{aligned}$$

In the second case, we write $\mathbf{A} = A, \mathbf{B} = BE, \mathbf{C} = CE, \mathbf{D} = D$ and find

$$\begin{aligned}
&\tfrac{1}{2}(\mathbf{A}+\mathbf{D}) + \tfrac{1}{2}(\mathbf{A}-\mathbf{D})E + \mathbf{C}n_o + \mathbf{B}(-n_\infty/2) = \\
&\quad = \tfrac{1}{2}(A+D) + \tfrac{1}{2}(A-D)E + CE\,n_o + BE\,(-n_\infty/2)\\
&\quad = \tfrac{1}{2}(A+D) + \tfrac{1}{2}(A-D)E - Cn_o + B(-n_\infty/2)\\
&\quad = \tfrac{1}{2}(A+\widehat{D}) + \tfrac{1}{2}(A-\widehat{D})E + \widehat{C}n_o + B(-n_\infty/2).
\end{aligned}$$

In both cases, therefore, the interpretation eq.(6) is confirmed.

6 CONCLUSION

We have spelled out how conformal geometric algebra can be represented by matrices – Vahlen matrices for the versors, and extended Vahlen matrices (with possibly zero determinant) for the blades. We have found this representation useful for the derivation of some reasonably advanced but mostly known results.

A caveat: the conformal model is subject to a US patent [5].

REFERENCES

[1] P. Anglès, *Construction de revêtements du group conform d'un espace vectorial muni d'une "métrique" de type (p,q)*. Annales de l'Institut Henri Poincaré, Section A, Vol. XXXIII:33-51, 1980

[2] C. Cibura, L. Dorst, *From Exact Correspondence Data to Conformal Transformations in Closed Form Using Vahlen Matrices*, Gravisma 2009.

[3] L. Dorst, D. Fontijne, S. Mann, *Geometric Algebra for Computer Science*, Morgan Kaufmann, 2007.

[4] D. Hestenes, *The Design of Linear Algebra and Geometry*, Acta Applicandae Mathematicae, Kluwer Academic Publishers 23: 65-93, 1991.

[5] D. Hestenes, A. Rockwood, H. Li, *System for encoding and manipulating models of objects*, U.S. Patent 6,853,964, granted February 8, 2005

[6] D. Hestenes, *New tools for Computational Geometry and Rejuvenation of Screw Theory*, AGACSE 2008, to be published 2009.

[7] P. Lounesto, *Clifford Algebras and Spinors*, LMSLNS 286,Cambridge University Press, 2001.

Using Geometric Algebra for Navigation in Riemannian and Hard Disc Space

Werner Benger
Center for Computation & Technology
Louisiana State University
239 Johnston Hall
Baton Rouge, LA 70803, USA
werner@cct.lsu.edu

Andrew Hamilton
Center for Astrophysics and Space Astronomy
JILA, University of Colorado
Boulder, CO 80309, USA
Andrew.Hamilton@colorado.edu

Mike Folk/Quincey Koziol
The HDF Group
1901 So. First St. Suite C-2
Champaign, IL 61820. USA
mfolk@hdfgroup.org

Simon Su
Princeton Institute for Computational Science and Engineering
345 Peter B. Lewis Library
Princeton, NJ 08544, USA
simonsu@princeton.edu

Erik Schnetter
Center for Computation & Technology
Dept. of Physics & Astronomy
Louisiana State University
Baton Rouge, LA 70803, USA
schnetter@cct.lsu.edu

Marcel Ritter/Georg Ritter
Department of Computer Science
University of Innsbruck
Technikerstrasse 21a
A-6020 Innsbruck, Austria
csab7885@uibk.ac.at

ABSTRACT

A "vector" in 3D computer graphics is commonly understood as a triplet of three floating point numbers, eventually equipped with a set of functions operating on them. This hides the fact that there are actually different kinds of vectors, each of them with different algebraic properties and consequently different sets of functions. Differential Geometry (DG) and Geometric Algebra (GA) are the appropriate mathematical theories to describe these different types of "vectors". They consistently define the proper set of operations attached to each class of "floating point triplet" and allow to derive what meta-information is required to uniquely identify a specific type of vector in addition to its purely numerical values. We shortly review the various types of "vectors" in 3D computer graphics, their relations to rotations and quaternions, and connect these to the terminology of co-vectors and bi-vectors in DG and GA. Not only in 3D, but also in 4D, the elegant formulations of GA yield to more clarity, which will be demonstrated on behalf of the use of bi-quaternions in relativity, allowing for instance a more insightful formulation to determine the Newman-Penrose pseudo scalars from the Weyl tensor.

1. INTRODUCTION

Geometric Algebra [7] and the sometimes mystified concept of spinors eases implementation and intuition significantly, both in computer graphics and in physics [8]. GA provides means to describe how the metadata information required per "vector" can be provided in persistent storage. Given large datasets that are expensively collected or generated by simulations requiring millions of CPU hours, it is increasingly important and difficult to be able to share and correctly interpret such datasets years after their generation, across different research groups from different fields of science. A unique, standardized, extensible identification of the geometric properties of the dataset elements is a necessary pre-requisite for this. similar to the way in which the IEEE standard for floating point values enables sharing floating point values. The HDF5 library here, a generic self-describing file format developed for large datasets as originating in high performance computing, provides useful means: It allows specifying metadata in addition to the purely numerical data, thereby enabling an abstraction layer for specifying the mathematical properties on top of the lower-level binary layout. In this article we describe a how to use the functionality of this powerful I/O library to express the semantics of vector quantities as they arise in Geometric Algebra, as required by complex applications such as general relativity exemplified by gravitational wave astronomy.

2. VECTOR SPACES

A vector space over a field F (such as $\mathbb{R}$) is a set $\mathcal{V}$ together with two binary operations *vector addition* $+ : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ and *scalar multiplication* $\circ : F \times \mathcal{V} \to \mathcal{V}$. The elements of $\mathcal{V}$ are called *vectors*. A vector space is closed under the operations $+$ and $\circ$, i.e., for all elements $u, v \in \mathcal{V}$ and all elements $\lambda \in F$ there is $u + v \in \mathcal{V}$ and $\lambda \circ u \in \mathcal{V}$ (vector space axioms). The vector space axioms allow computing the differences of vectors and therefore defining the derivative of a vector-valued function $v(s) : \mathbb{R} \to \mathcal{V}$ as

$$\frac{d}{ds}v(s) := \lim_{ds \to 0} \frac{v(s+ds) - v(s)}{ds} \quad . \tag{1}$$

2.1 Tangential Vectors

In differential geometry, a tangential vector on a manifold M is the operator $\frac{d}{ds}$ that computes the derivative along a curve $q(s) : \mathbb{R} \to M$ for an arbitrary scalar-valued function

$f : M \to \mathbb{R}$:

$$\frac{d}{ds} f \Big|_{q(s)} := \frac{df\,(q(s))}{ds} \quad . \tag{2}$$

Tangential vectors fulfill the vector space axioms and can therefore be expressed as linear combinations of derivatives along the n coordinate functions $x^\mu : M \to \mathbb{R}$ with $\mu = 0 \dots n-1$, which define a basis of the tangential space $T_{q(s)}(M)$ on the n-dimensional manifold M at each point $q(s) \in M$:

$$\frac{d}{ds} f = \sum_{\mu=0}^{n-1} \frac{dx^\mu\,(q(s))}{ds} \frac{\partial}{\partial x^\mu} f =: \sum_{\mu=0}^{n-1} \dot{q}^\mu \partial_\mu f \tag{3}$$

where $\dot{q}^\mu$ are the components of the tangential vector $\frac{d}{ds}$ in the chart $\{x^\mu\}$ and $\{\partial_\mu\}$ are the basis vectors of the tangential space in this chart. We will use the Einstein sum convention in the following text, which assumes implicit summation over indices occurring on the same side of an equation. Often tangential vectors are used synonymous with the term "vectors" in computer graphics when a direction vector from point A to point B is meant. A tangential vector on an n-dimensional manifold is represented by n numbers in a chart.

2.2 Co-Vectors

The set of operations $df : T(M) \to \mathbb{R}$ that map tangential vectors $v \in T(M)$ to a scalar value $v(f)$ for any function $f : M \to \mathbb{R}$ defines another vector space which is dual to the tangential vectors. Its elements are called *co-vectors*.

$$< df, v >= df(v) := v(f) = v^\mu \partial_\mu f = v^\mu \frac{\partial f}{\partial x^\mu} \tag{4}$$

Co-vectors fulfill the vector space axioms and can be written as linear combination of co-vector basis functions dx^μ:

$$df =: \frac{\partial f}{\partial x^\mu} dx^\mu \tag{5}$$

with the dual basis vectors fulfilling the duality relation

$$< dx^\nu, \partial_\mu >= \begin{cases} \mu = \nu : & 1 \\ \mu \neq \nu : & 0 \end{cases} \tag{6}$$

The space of co-vectors is called the co-tangential space $T^*_p(M)$. A co-vector on an n-dimensional manifold is represented by n numbers in a chart, same as a tangential vector. However, co-vector transforms inverse to tangential vectors when changing coordinate systems, as is directly obvious from eq. (6) in the one-dimensional case: As $< dx^0, \partial_0 >= 1$ must be sustained under coordinate transformation, dx^0 must shrink by the same amount as ∂_0 grows when another coordinate scale is used to represent these vectors. In higher dimensions this is expressed by an inverse transformation matrix, as demonstrated in Fig. 1. In Euclidean three-dimensional space, a plane is equivalently described by a "normal vector", which is orthogonal to the plane. While "normal vectors" are frequently symbolized via a vector arrow, like tangential vectors, they are not the same, rather they are dual to tangential vectors. It is more appropriate to visually symbolize them as a plane. This visual is also supported by (5), which can be interpreted as the total differential of a function f: a co-vector describes how a scalar function advances in space, which can be visualized as surfaces of constant function value ("isosurface"). On an n-dimensional manifold a co-vector is correspondingly symbolized by an $(n-1)$-dimensional subspace.

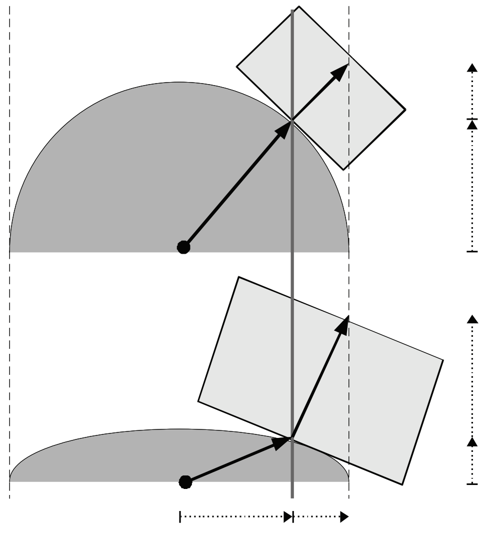


Figure 1: Vector transformation under shrinking the height coordinate by a factor of two: tangential vectors (differences between two points) shrink in their height component by a factor two as well, whereas surface normal vectors (co-vectors) grow by a factor two in height, see the vertical components of the vector and co-vector shown on the right hand side in the figure.

2.3 Tensors

A *tensor* T^l_m of rank $l \times m$ is a multi-linear map of l vectors and m co-vectors to a scalar

$$T^l_m : \underbrace{T(M) \times \cdots \times T(M)}_{l} \times \underbrace{T^*(M) \times \cdots \times T^*(M)}_{m} \to \mathbb{R} \quad . \tag{7}$$

Tensors are elements of a vector space themselves and form the tensor algebra. They are represented relative to a coordinate system by a set of k^{l+m} numbers for a k-dimensional manifold. The construction of an tensor of higher rank from lower rank is called the *outer product* (also known as tensor, dyadic or Kronecker product), denoted by $\otimes$:

$$T \equiv T^{\mu\nu} \partial_\mu \otimes \partial_\nu = v^\mu u^\nu \partial_\mu \otimes \partial_\nu = v^\mu \partial_\mu \otimes u^\nu \partial_\nu = v \otimes u \tag{8}$$

Tensors of rank 2 may be represented using matrix notation. Tensors of type T^0_1 are equivalent to co-vectors and called co-variant, in matrix notation (relative to a chart) they correspond to rows. Tensors of type T^1_0 are equivalent to a tangential vector and are called contra-variant, corresponding to columns in matrix notation. The duality relationship between vectors and co-vectors then corresponds to the matrix multiplication of a $1 \times n$ row with a $n \times 1$ column, yielding a single number

$$< a, b >=< a^\mu \partial_\mu, b_\mu dx^\mu > \equiv (a^0 \dots a^{n-1}) \begin{pmatrix} b^0 \\ \dots \\ b^{n-1} \end{pmatrix} \quad . \tag{9}$$

By virtue of the duality relationship (6) the contraction of lower and upper indices is defined as the *interior product* ι of tensors, which reduces the dimensionality of the tensor:

$$\iota : T^m_n \times T^l_k \to T^{m-k}_{n-l} : (u, v) \mapsto \iota_u v \tag{10}$$

The interior product can be understood (visually) as a generalization of some "projection" of a tensor onto another one.

Of special importance are symmetric tensors of rank two $g \in T^0_2$ with $g : T(M) \times T(M) \to \mathbb{R} : u, v \mapsto g(u, v) \equiv u \cdot v$, $g(u, v) = g(v, u)$, as they can be used to define a *metric* on the tangential vectors, also called the *inner product* or dot product. Its inverse, defined by operating on the co-vectors, is called the co-metric. A metric, same as the co-metric, is represented as a symmetric $n \times n$ matrix in a chart for a n-dimensional manifold.

Given a metric tensor, one can define equivalence relationships between tangential vectors and co-vectors, which allow to map one into each other. These maps are called the "musical isomorphisms", $\flat$ and $\sharp$, as they raise or lower an index in the coordinate representation:

$$\flat : \quad T(M) \to T^*(M) : \quad v^\mu \partial_\mu \mapsto v^\mu g_{\mu\nu} dx^\nu \qquad (11)$$

$$\sharp : \quad T^*(M) \to T(M) : \quad V_\mu dx^\mu \mapsto V_\mu g^{\mu\nu} \partial_\nu \qquad (12)$$

As an example application, the "gradient" of a scalar function is given by $\nabla f = \sharp df$ using this notation. In Euclidean space, the metric is represented by the identity matrix and the components of vectors are identical to the components of co-vectors. As computer graphics usually is considered in Euclidean space, this justifies the usual negligence of distinction among vectors and co-vectors; consequently graphics software only knows about one type of vectors which is uniquely identified by its number of components. However, when dealing with coordinate transformations or curvilinear mesh types then distinguishing between tangential vectors and co-vectors is unavoidable. Treating them both as the same type within a computer program leads to confusions and is not safe. Section 4 will address this issue.

2.4 Exterior Product

The *exterior product* $\wedge : \mathcal{V} \times \mathcal{V} \to \Lambda^2(\mathcal{V})$ (also known as wedge product, Grassmann product, or alternating product) generates vector space elements of higher dimensions from elements of a vector space $\mathcal{V}$ by taking the antisymmetric part of the outer product (eq. 8) as

$$u \wedge v = \frac{1}{2} (u \otimes v - v \otimes u) \qquad (13)$$

The new vector space is denoted $\Lambda^2(\mathcal{V})$. With the exterior product, $v \wedge u = -u \wedge v \quad \forall u, v \in V$, which consequently results in $v \wedge v = 0 \; \forall \; v \in \mathcal{V}$. The exterior product defines an algebra on its elements, the exterior algebra (or Grassman algebra) [2]. It is a sub-algebra of the Tensor algebra consisting on the anti-symmetric tensors. The exterior algebra is defined intrinsically by the vector space and does not require a metric. For a given n-dimensional vector space $\mathcal{V}$, there can at most be n-th power of an exterior product, consisting of n different basis vectors. The $n + 1$-th power must vanish, because at least one basis vector would occur twice, and there is exactly one basis vector for $\Lambda^n(\mathcal{V})$.

Elements $v \in \Lambda^k(\mathcal{V})$ are called k-vectors, whereby 2-vectors are also called bi-vectors and 3-vectors trivectors. The number of components of an k-vector of an n-dimensional vector space is given by the binomial coefficient $\binom{n}{k}$. For $n = 2$ there are two 1-vectors and one bi-vector, for $n = 3$ there are three 1-vectors, three bi-vectors and one tri-vector. These relationships are depicted by the Pascal's triangle, with the

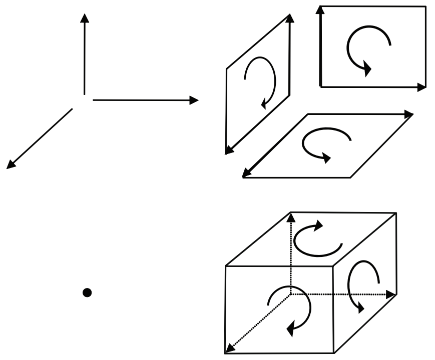


Figure 2: Graphical representation of the $1+3+3+1$ structure of components that build a 3D multivector: three tangential vectors, three oriented planes, one scalar and one (oriented) volume element.

row representing the dimensionality of the underlying base space and the column the vector type:

$$\begin{array}{ccccccccc} &&&& 1 &&&& \\ &&& 1 && 1 &&& \\ && 1 && 2 && 1 && \\ & 1 && 3 && 3 && 1 & \\ 1 && 4 && 6 && 4 && 1 \end{array} \qquad (14)$$

As can be easily read off, for a four-dimensional vector space there will be four 1-vectors, six bi-vectors, four tri-vectors and one 4-vector. The n-vector of a n-dimensional vector space is also called a *pseudo-scalar*, the $(n - 1)$ vector a *pseudo-vector*.

2.5 Visualizing Exterior Products

An exterior algebra is defined on both the tangential vectors and co-vectors on a manifold. A bi-vector v formed from tangential vectors $\{\partial_\mu\}$ and a bi-covector U formed from co-vectors $\{dx^\mu\}$ are written in a chart as

$$v = v^{\mu\nu} \partial_\mu \wedge \partial_\nu \qquad , \quad U = U_{\mu\nu} dx^\mu \wedge dx^\nu \quad . \qquad (15)$$

They both have $\binom{n}{2}$ independent components, due to $v^{\mu\nu} = -v^{\nu\mu}$ and $U_{\mu\nu} = -U_{\nu\mu}$ (three components in 3D, six components in 4D). A bi-tangential vector can be understood visually as an (oriented, i.e., signed) plane that is spun by the two defining tangential vectors, independently of the dimensionality of the underlying base space. A bi-co-vector corresponds to the subspace of an n-dimensional hyperspace where a plane is "cut out". In three dimensions these visualizations overlap: both a bi-tangential vector and a co-vector correspond to a plane, and both a tangential vector and a bi-co-vector correspond to one-dimensional direction ("arrow"). In four dimensions, these visuals are more distinct but still overlap: a co-vector corresponds to a three-dimensional volume, but a bi-tangential vector is represented by a plane similar to a bi-co-vector, since cutting out a 2D plane from four-dimensional space yields a 2D plane again. Only in higher dimensions these symbolic representations become unique.

However, in any case a co-vector and a pseudo-vector will have the same appearance as an $n - 1$ dimensional hyperspace, same as a tangential vector corresponds to an pseudo-

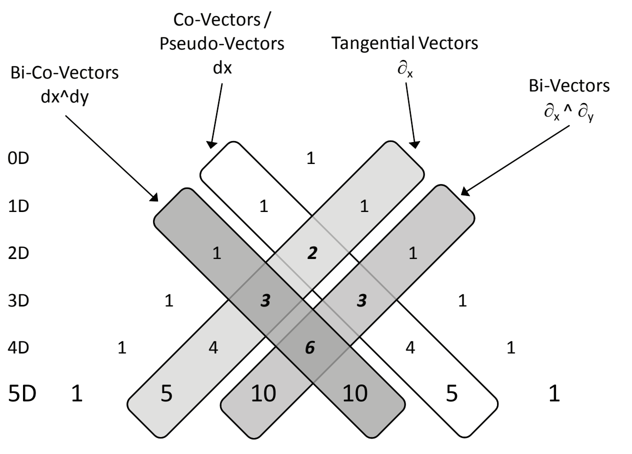


Figure 3: Pascal's triangle showing the location of tangential vectors, bi-vectors, co-vectors and bi-covectors in the various subspaces in different dimensions. Especially in three dimensions there are many overlaps, indicating ambiguities where different quantities are all represented by "just three numbers". Similar situations occur in 4D, only in 5D all vector types become unambiguous.

co-vector:

$$V_\mu dx^\mu \quad\Leftrightarrow\quad v^{\alpha_0\alpha_1\ldots\alpha_{n-1}}\,\partial_{\alpha_0}\wedge\partial_{\alpha_1}\wedge\ldots\partial_{\alpha_{n-1}} \tag{16}$$
$$v^\mu\partial_\mu \quad\Leftrightarrow\quad V_{\alpha_0\alpha_1\ldots\alpha_{n-1}}\,dx^{\alpha_0}\wedge dx^{\alpha_1}\wedge\ldots dx^{\alpha_{n-1}} \tag{17}$$

A tangential vector – lhs of (17) – can be understood as one specific direction, but equivalently as well as "cutting off" all but one $n-1$-dimensional hyperspaces from an n-dimensional hyperspace – rhs of (17). This equivalence is expressed via the interior product of a tangential vector v with an pseudo-co-scalar Ω yielding a pseudo-co-vector V (18), similarly the interior product of a pseudo-vector with an pseudo-co-scalar yielding a tangential vector (18):

$$\iota_\Omega : T(M) \;\to\; (T^*)^{n-1}(M) : V \mapsto \iota_\Omega v \tag{18}$$
$$\iota_\Omega : T^{n-1}(M) \;\to\; T^*(M) : V \mapsto \iota_\Omega v \tag{19}$$

Pseudo-scalars and pseudo-co-scalars will always be scalar multiples of the basis vectors $\partial_{\alpha_0}\wedge\partial_{\alpha_1}\wedge\ldots\partial_{\alpha_n}$ and $dx^{\alpha_0}\wedge dx^{\alpha_1}\wedge\ldots dx^{\alpha_n}$. However, under when inversing a coordinate $x^\mu\to -x^\mu$ they flip sign, whereas a "true" scalar does not. An example known from Euclidean vector algebra is the allegedly scalar value constructed from the dot and cross product of three vectors $V(u,v,w)=u\cdot(v\times w)$ which is the negative of when its arguments are flipped:

$$V(u,v,w) = -V(-u,-v,-w) = -u\cdot(-v\times -w) \quad . \tag{20}$$

This property is obvious when written as exterior product:

$$V(u,v,w) = u\wedge v\wedge w = V\partial_0\wedge\partial_1\wedge\partial_2 \tag{21}$$

This expression actually describes a multiple of a volume element spun by the basis tangential vectors ∂_μ - any volume must be a scalar multiple of this basis volume element, but can flip sign if another convention on the basis vectors is used. This convention - right-handed versus left-handed coordinate system - is expressed by the orientation tensor $\Omega = \pm\partial_0\wedge\partial_1\wedge\partial_2$. In computer graphics both conventions occur, which often causes confusion.

By combining (19) and (12) – requiring a metric – we get a map from pseudo-vectors to vectors and reverse. This map is known as the *Hodge star operator* "$\star$":

$$\star : T^{n-1}(M) \to T(M) : V \longmapsto \sharp\iota_\Omega V \tag{22}$$

The same operation can be applied to the co-vectors accordingly, and generalized to all vector elements of the exterior algebra on a vector space, establishing a correspondence between $k-vectors$ and $n-k$-vectors. The Hodge star operator allows to identify vectors and pseudo-vectors, similarly to how a metric allows to identify vectors and co-vectors. The Hodge star operator requires a metric and an orientation Ω.

2.6 Geometric Algebra

Geometric Algebra postulates a product on elements of a vector space $u,v,w\in\mathcal{V}$ that is associative, $(uv)w=u(vw)$, left-distributive $u(v+w)=uv+uw$, right-distributive $(u+v)w=uw+vw$, and reduces to the inner product as defined by the metric $v^2=g(v,v)$. It can be shown that the sum of the exterior product (which is also called "outer product" within GA, but should not be confused with the outer product $\otimes$ on tensors from eq. 8) and the inner product fulfill these requirements; this defines the *geometric product*:

$$uv := u\wedge v + u\cdot v \quad . \tag{23}$$

Since $u\wedge v$ and $u\cdot v$ are of different dimensionality ($\binom{n}{2}$ and $\binom{n}{0}$, respectively), the result must be in a higher dimensional vector space of dimensionality $\binom{n}{2}+\binom{n}{0}$. This space, called $\Lambda(\mathcal{V})$, is formed by the linear combination of k-vectors:

$$\Lambda(\mathcal{V}) = \bigoplus_{k=0}^{n}\Lambda^k(\mathcal{V}) \quad . \tag{24}$$

Its elements are called *multivectors*. The dimensionality of $\Lambda(\mathcal{V})$ is $\sum_{k=0}^n\binom{n}{k}\equiv 2^n$.

For instance, in two dimensions the dimension of the space of multivectors is $2^2=4$. A multivector V, constructed from tangential-vectors on a 2D manifold, is written as

$$V = V^0 + V^1\partial_0 + V^2\partial_1 + V^3\partial_0\wedge\partial_1 \tag{25}$$

with V^μ the four components of the multivector in a chart. For a three-dimensional manifold a multivector on its tangential space has $2^3=8$ components and is written as

$$\begin{aligned} V =& V^0 + \\ & V^1\partial_0 + V^2\partial_1 + V^2\partial_2 + \\ & V^4\partial_0\wedge\partial_1 + V^5\partial_1\wedge\partial_2 + V^6\partial_2\wedge\partial_0 + \\ & V^7\partial_0\wedge\partial_1\wedge\partial_2 \end{aligned} \tag{26}$$

with V^μ the eight components of the multivector in a chart. The components of a multivector have a direct visual interpretation, which is one of the key features of geometric algebra. In 3D, a multivector is the sum of a scalar value, three directions, three planes and one volume. These basis elements span the entire space of multivectors.

3. NEWMAN-PENROSE FORMALISM

General relativity predicts the existence of gravitational waves. There is a huge effort to detect gravitational waves expected for example from merging pairs of black holes.To date no gravitational waves have been detected directly. There is however indirect evidence for their existence from

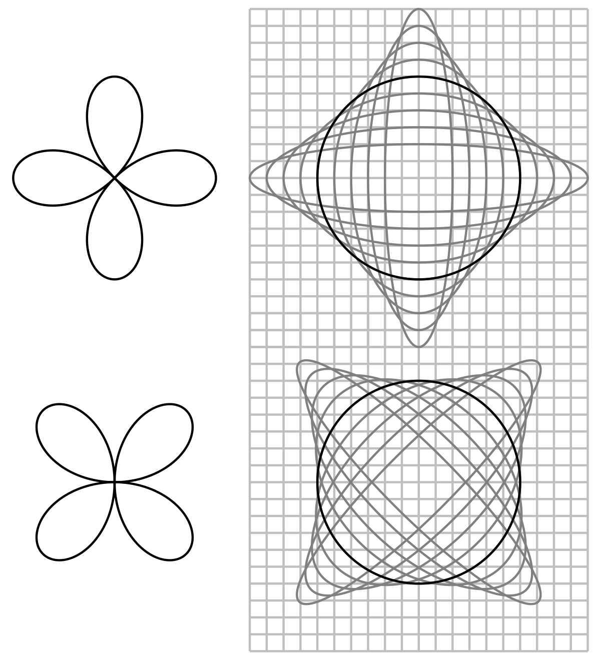


Figure 4: The two linear polarizations of gravitational waves. The + polarization (top) has a $\cos 2\chi$ shape about the direction of propagation (into the paper), while the $\times$ polarization (bottom) has a $\sin 2\chi$ shape. A gravitational wave causes a system of freely falling test masses to oscillate relative to a grid of points a fixed proper distance apart.

the gradual decrease in orbital period of the binary pulsar[1], which is quantitatively consistent with the general relativistic prediction of energy loss by quadrupole emission of gravitational waves.

It is conventional to characterize gravitational waves in terms of their Newman-Penrose (1962) (NP) components [9, 11]. The purpose of this section is to give an idea of how this works, and how the geometric algebra offers insight into the NP formalism. The traditional derivation of the NP components of gravitational waves is magical, and shrouded in unnecessary and misleading notation. As Held (1974) [6] politely puts it, the NP formalism presents "a formidable notational barrier to the uninitiate".

The notion of a gravitational wave can be perplexing. A passing gravitational wave causes the distance between two freeling-falling masses to oscillate. But if gravity affects the very measurement of length itself, how can the distance between the masses be measured? The answer is that, despite the fact that in general relativity spacetime has no absolute existence, in the sense that the choice of coordinate system is arbitrary, nevertheless the metric asserts that there is a unique proper distance along a given path (or affine distance, along a null path) between any two points in spacetime (such as the path followed by a beam of laser light). The presence of gravity, or curvature, is expressed by the presence of a gravitational force between two points a fixed proper distance apart. A gravitational wave causes an oscillation in the differential gravitational force, or tidal force, between two points a fixed distance apart.

Figure 4 illustrates gravitational waves, in their two possible linear polarizations, + and $\times$. The grid represents a locally inertial system of points a fixed proper distance apart. The superposed ellipses represent a system of freely-falling test masses whose positions, initially on a circle, are being perturbed by a gravitational wave moving in a direction perpendicular to the paper. The proper distance between freely-falling test masses oscillates. That oscillation can be measured for example by the change in the number of wavelengths along a laser beam between the masses.

3.1 Newman-Penrose tetrad

The Newman-Penrose (NP) formalism is particularly well adapted to treating waves that travel at the speed of light, which includes electromagnetic and gravitational waves. The NP formalism starts with the rest frame of an observer, and applies two tricks to it. The axes, or tetrad, of the observer's locally inertial frame form an orthonormal basis of vectors in the geometric algebra $\{\boldsymbol{\gamma}_t, \boldsymbol{\gamma}_x, \boldsymbol{\gamma}_y, \boldsymbol{\gamma}_z\}$, with the metric in Minkowski signature of the form

$$\boldsymbol{\gamma}_m \cdot \boldsymbol{\gamma}_n = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{27}$$

with indices m, n running over t, x, y, z. The NP formalism chooses one axis, typically the z-axis, to be the direction of propagation of the wave.

The first NP trick is to replace the transverse axes $\boldsymbol{\gamma}_x$ and $\boldsymbol{\gamma}_y$ by spinor axes $\boldsymbol{\gamma}_+$ and $\boldsymbol{\gamma}_-$ defined by

$$\boldsymbol{\gamma}_+ \equiv \frac{1}{\sqrt{2}} (\boldsymbol{\gamma}_x + I \boldsymbol{\gamma}_y) \ , \quad \boldsymbol{\gamma}_- \equiv \frac{1}{\sqrt{2}} (\boldsymbol{\gamma}_x - I \boldsymbol{\gamma}_y) \ . \tag{28}$$

This is the same trick used to define the spinor components $L_\pm$ of the angular momentum operator $\boldsymbol{L}$ in quantum mechanics.

The second NP trick is to replace the time t and propagation z axes with outgoing and ingoing null axes $\boldsymbol{\gamma}_v$ and $\boldsymbol{\gamma}_u$, defined by

$$\boldsymbol{\gamma}_v \equiv \frac{1}{\sqrt{2}} (\boldsymbol{\gamma}_t + \boldsymbol{\gamma}_z) \ , \quad \boldsymbol{\gamma}_u \equiv \frac{1}{\sqrt{2}} (\boldsymbol{\gamma}_t - \boldsymbol{\gamma}_z) \ . \tag{29}$$

The resulting outgoing, ingoing, and spinor axes form a NP null tetrad

$$\{\boldsymbol{\gamma}_v, \boldsymbol{\gamma}_u, \boldsymbol{\gamma}_+, \boldsymbol{\gamma}_-\} \ , \tag{30}$$

with NP metric

$$\boldsymbol{\gamma}_m \cdot \boldsymbol{\gamma}_n = \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{31}$$

with indices m, n running over $v, u, +, -$. The NP metric (31) has zeros down the diagonal. This means that each of the four NP axes $\boldsymbol{\gamma}_m$ is null: the scalar product of each

[1]`http://nobelprize.org/nobel_prizes/physics/laureates/1993/illpres/discovery.html`

axis with itself is zero. In a profound sense, the null, or light-like, character of each the four NP axes explains why the NP formalism is well adapted to treating fields that propagate at the speed of light.

Three kinds of transformation, considered further below, take a particularly simple form in the NP tetrad:

I Reflections through the transverse axis y;
II Rotations about the propagation axis z;
III Boosts along the propagation axis z.

3.1.1 Reflections

Under transformation I, a reflection through the y-axis, the spinor axes swap:

$$\gamma_+ \leftrightarrow \gamma_- \ , \tag{32}$$

which may also be accomplished by complex conjugation. Reflection through the y-axis, or equivalently complex conjugation, changes the sign of all spinor indices of a tensor component

$$+ \leftrightarrow - \ . \tag{33}$$

In short, complex conjugation flips spin, a pretty feature of the NP formalism.

3.1.2 Rotations

Under transformation II, a right-handed rotation by angle χ about the direction z of propagation, the transverse axes γ_x and γ_y transform as

$$\begin{aligned} \gamma_x &\rightarrow \cos\chi\,\gamma_x - \sin\chi\,\gamma_y \ , \\ \gamma_y &\rightarrow \sin\chi\,\gamma_x + \cos\chi\,\gamma_y \ . \end{aligned} \tag{34}$$

It follows that the spinor axes γ_+ and γ_- transform under a right-handed rotation by angle χ as

$$\gamma_\pm \rightarrow e^{\pm I\chi}\,\gamma_\pm \ . \tag{35}$$

The transformation (35) identifies the spinor axes γ_+ and γ_- as having spin $+1$ and -1 respectively. More generally, an object can be defined as having spin s if it varies by

$$e^{sI\chi} \tag{36}$$

under a rotation by angle χ about the direction of propagation. The NP components of a tensor inherit spin properties from that of the spinor basis. The general rule is that the spin s of any tensor component is equal to the number of $+$ covariant indices minus the number of $-$ covariant indices:

$$\text{spin } s = \text{number of } + \text{ minus } - \text{ covariant indices} \ . \tag{37}$$

3.1.3 Boosts

The final transformation III, a boost along the z-axis, multiplies the outgoing and ingoing axes γ_v and γ_u by a blueshift factor ϵ and its reciprocal

$$\begin{aligned} \gamma_v &\rightarrow \epsilon\,\gamma_v \ , \\ \gamma_u &\rightarrow (1/\epsilon)\,\gamma_u \ . \end{aligned} \tag{38}$$

If the observer boosts by velocity v in the z-direction away from the source, then the blueshift factor is the special relativistic Doppler shift factor

$$\epsilon = \left(\frac{1-v}{1+v}\right)^{1/2} \ . \tag{39}$$

The exponent n of the power ϵ^n by which an object changes under a boost along the z-axis is called its boost weight. Thus γ_v has boost weight $+1$, and γ_u has boost weight -1. The NP components of a tensor inherit their boost weight properties from those of the NP basis. The general rule is that the boost weight n of any tensor component is equal to the number of v covariant indices minus the number of u covariant indices:

$$\text{boost weight } n = \text{number of } v \text{ minus } u \text{ covariant indices} \ . \tag{40}$$

3.2 Electromagnetic waves

The properties of gravitational waves are in many ways similar to those of electromagnetic waves. Both kinds of waves are massless, traveling at the speed of light. A crucial difference is that gravitational waves are spin-2 (tensor) waves, whereas electromagnetic waves are spin-1 (vector) waves.

Recall the nature of electromagnetic waves. Electromagnetic waves are characterized by the electromagnetic field F_{ij}, which is an antisymmetric tensor, or bivector, with 6 distinct components. The 6 components are commonly collected into two 3-dimensional vectors, the electric and magnetic fields E and B. The geometric algebra gives the insight that the electromagnetic field tensor, being a bivector, has a natural complex structure, in which the electric and magnetic fields together form a complex 3-vector $E + IB$.

With respect to a NP null tetrad (30), the electromagnetic bivector has 3 complex components, of spin respectively -1, 0, and $+1$, in accordance with the rule (37):

$$\begin{aligned} -1 &: F_{u-} \\ 0 &: \frac{1}{2}(F_{uv} + F_{+-}) \\ +1 &: F_{v+} \ . \end{aligned} \tag{41}$$

The complex conjugates of the 3 components are:

$$\begin{aligned} -1^* &: F_{u+} \\ 0^* &: \frac{1}{2}(F_{uv} - F_{+-}) \\ +1^* &: F_{v-} \ , \end{aligned} \tag{42}$$

whose spins have the opposite sign. Conventionally (Chandrasekhar 1983), the 3 complex spin components of the electromagnetic field bivector in the NP formalism are denoted

$$\begin{aligned} -1 &: \phi_2 \ , \\ 0 &: \phi_1 \ , \\ +1 &: \phi_0 \ . \end{aligned} \tag{43}$$

For outgoing electromagnetic waves, only the spin -1 component propagates, carrying electromagnetic energy far away from a source:

$$-1 \ : \ \text{propagating, outgoing} \ . \tag{44}$$

This propagating, outgoing -1 component has spin -1, but its complex conjugate has spin $+1$, so effectively both spin components, or helicities, of an outgoing wave are embodied in the single complex component. The remaining 2 complex NP components (spins 0 and 1) of an outgoing wave are short range, describing the electromagnetic field near the source. Similarly, for ingoing waves, only the spin $+1$ component propagates. The isolation of each propagating mode into a single complex NP mode, incorporating both helicities, is simpler than the standard picture of oscillating orthogonal electric and magnetic fields.

3.3 Gravitational waves

In electromagnetism, the electromagnetic field tensor is defined by the commutator of the gauge-covariant derivative. In general relativity, the analogous commutator of the covariant derivative is the Riemann curvature tensor R_{klmn}. The Riemann curvature tensor has symmetries which can be designated shorthandly

$$R_{([kl][mn])} \ . \tag{45}$$

Here [] denotes antisymmetry, and () symmetry. The designation (45) thus signifies that the Riemann curvature tensor R_{klmn} is antisymmetric in its first two indices kl, antisymmetric in its last two indices mn, and symmetric under exchange of the first and last pairs of indices, $kl \leftrightarrow mn$. In addition to the symmetries (45), the Riemann curvature tensor has the totally antisymmetric symmetry

$$R_{klmn} + R_{kmnl} + R_{knlm} = 0 \ . \tag{46}$$

The symmetries (45) imply that that the Riemann curvature tensor is a symmetric matrix of antisymmetric tensors, which is to say, a 6×6 symmetric matrix of bivectors. A 6×6 symmetric matrix has 21 independent components. The additional condition (46) eliminates one degree of freedom, leaving the Riemann curvature tensor with 20 independent components.

In spacetime algebra any bivector U (6 component) can be written as complex sum $U = (E + IB)\gamma_t$ of two spatial 3-vectors $E = E^x\gamma_x + E^y\gamma_y + B^z\gamma_z$ and $B = B^x\gamma_x + B^y\gamma_y + B^z\gamma_z$, due to the identity $I\gamma_x\gamma_t \equiv \gamma_y\gamma_z$ etc. In analogy to electromagnetism, $E\gamma_t$ is called the electric bivector, $B\gamma_t$ the magnetic bivector. The Riemann tensor, a multilinear multimap on bivectors, can then be organized into a 2×2 matrix of 3×3 blocks with bivector indices, yielding the structure

$$\begin{pmatrix} R_{EE} & R_{EB} \\ R_{BE} & R_{BB} \end{pmatrix} \ . \tag{47}$$

The condition of being symmetric implies that R_{EE} and R_{BB} are symmetric, while $R_{BE} = (R_{EB})^{\top}$. The condition (46) states that the 3×3 block R_{EB} (and likewise R_{BE}) is traceless.

The natural complex structure of bivectors in the geometric algebra suggests recasting the 6×6 Riemann curvature matrix (47) into a 3×3 complex matrix, which would have the structure $(R_E + IR_B)(R_E + IR_B)$, or equivalently

$$R_{EE} - R_{BB} + I(R_{EB} + R_{BE}) \ , \tag{48}$$

which is a complex linear combination of the four 3×3 blocks of the Riemann matrix (47). However, it turns out that the complex symmetric 3×3 matrix (48) encodes only part of the Riemann curvature tensor, namely the Weyl tensor. More specifically, the Riemann curvature tensor decomposes into a trace part, the Ricci tensor R_{km}, and a totally traceless part, the Weyl tensor C_{klmn}. The Ricci tensor, which is symmetric, has 10 independent components. The Weyl tensor, which inherits the symmetries (45) and (46) of the Rieman tensor, and in addition vanishes on contraction of any pair of indices, also has 10 independent components. Together, the Ricci and Weyl tensors account for the 20 components of the Riemann tensor. The components of the Ricci and Weyl tensors, though algebraically independent, are related by the differential Bianchi identities.

The end result is that the Weyl tensor, the traceless part of the Riemann curvature tensor, can be written as a 3×3 complex traceless symmetric matrix (48). Such a matrix has 5 distinct complex components.

In empty space (vanishing energy-momentum tensor), the Ricci tensor vanishes identically. Thus the properties of the gravitational field in empty space are specified entirely by the Weyl tensor. In particular, gravitational waves are specified entirely by the Weyl tensor.

When the 5 complex components of the Weyl tensor are expressed in a NP null tetrad (30), the result is 5 complex components, of spins respectively -2, -1, 0, $+1$, and $+2$:

$$\begin{aligned} -2 &: C_{u-u-} \\ -1 &: C_{uvu-} \\ 0 &: \frac{1}{2}(C_{vuvu} + C_{vu-+}) \\ +1 &: C_{vuv+} \qquad (49) \\ +2 &: C_{v+v+} \ . \qquad (50) \end{aligned}$$

It can be shown that these 5 complex components exhaust the degrees of freedom of the Weyl tensor.

For outgoing gravitational waves, only the spin -2 component propagates, carrying gravitational waves to far distances:

$$-2 \ : \ \text{propagating, outgoing} \ . \tag{51}$$

This propagating, outgoing -2 component has spin -2, but its complex conjugate has spin $+2$, so effectively both spin components, or helicities, or polarizations, of an outgoing wave gravitational wave are embodied in the single complex component. The remaining 4 complex NP components (spins -1 to 2) of an outgoing gravitational waves are short range, describing the gravitational field near the source.

Conventionally (Chandrasekhar 1983), the 5 complex spin components of the Weyl tensor in the NP formalism are impenetrably denoted

$$\begin{aligned} -2 &: \psi_4 \ , \\ -1 &: \psi_3 \ , \\ 0 &: \psi_2 \ , \\ +1 &: \psi_1 \\ +2 &: \psi_0 \ . \end{aligned} \tag{52}$$

Thus the component ψ_4 represents propagating, outgoing gravitational waves. The real part of ψ_4 represents the $\cos(2\chi)$, or $+$, polarization of the propagating gravitational wave, while (minus) its imaginary part represents the $\sin(2\chi)$, or $\times$, polarization, Figure 4. Next time you see an illustration of gravitational waves where the caption says that ψ_4 is plotted, that's what it is (see figure 5). We consider the formulation of the NP scalars as presented here much easier to understand than the usual approach, such as e.g. [11].

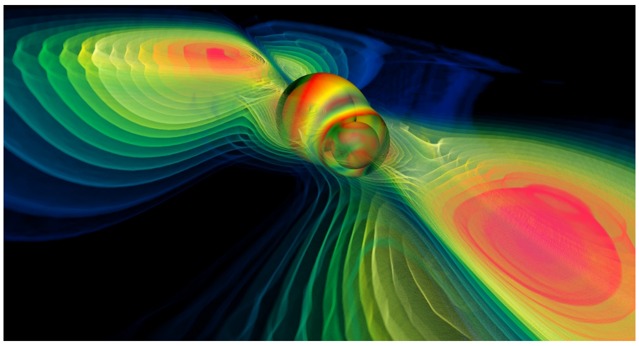


Figure 5: Volume rendering of the gravitational radiation during a binary black hole merger, represented by the real part of Weyl scalar $r \cdot \psi_4$.

4. IMPLEMENTING VECTORS IN C++

As demonstrated in section 2, denoting a vector by just its dimensionality n is insufficient to completely identify its algebraic properties including coordinate transformation rules. Additional information is needed, such as the number of covariant and contra-variance indices.

4.1 Class Hierarchy

Let us denote an array of fixed size N over some type T as `FixedArray<T,N>`, using C++ template notation. No algebraic operation shall be defined on this type, it just serves as a container for numbers, forming an N-tupel of T's. This definition serves as a base class for a type `Vector<T,N>`, which does not add new data members but only adds operators for addition of `Vector<T,N>`'s and multiplication with scalar values, yielding objects of type `Vector<T,N>` again.

$$\texttt{FixedArray<T,N>} \longrightarrow \texttt{Vector<T,N>} \tag{53}$$

The resulting class `Vector<T,N>` is a vector in the algebraic sense. It is convenient to make use of matrix algebra in many cases, and since matrices have vector space properties, to express such by deriving the `Matrix` class from the general `Vector` class:

$$\texttt{Vector<T,N*M>} \longrightarrow \texttt{Matrix<T,N,M>} \tag{54}$$

The matrix class will add the concept of a matrix product to the general vector space elements. A convenient, though not required, intermediate definition is to define rows and columns – they are rather type definitions than derived classes:

$$\texttt{Matrix<T,1,M>} \;\longrightarrow\; \texttt{Row<T,M>} \tag{55}$$
$$\texttt{Matrix<T,N,1>} \;\longrightarrow\; \texttt{Column<T,N>} \tag{56}$$

These definitions provide the basis for vector types to be used on the tangential space of a manifold. For a given N,T the following classes are derived:

$$\texttt{FixedArray<T,N>} \;\longrightarrow\; \texttt{point} \tag{57}$$
$$\texttt{Row<T,N>} \;\longrightarrow\; \texttt{covector} \tag{58}$$
$$\texttt{Column<T,N>} \;\longrightarrow\; \texttt{tvector} \tag{59}$$
$$\texttt{Vector} < \texttt{T}, \texttt{N}^2 - \texttt{N}(\texttt{N}+1)/2 > \;\longrightarrow\; \texttt{bivector} \tag{60}$$
$$\texttt{Vector} < \texttt{T}, 1 + \texttt{N}^2 - \texttt{N}(\texttt{N}+1)/2 > \;\longrightarrow\; \texttt{rotor} \tag{61}$$
$$\texttt{Vector} < \texttt{T}, 2^{\texttt{N}} > \;\longrightarrow\; \texttt{mulvector} \tag{62}$$

The definition of (58) and (59) directly implements the duality relationship (6) in a type-safe way. Tangential vectors and co-vectors both have vector space properties by virtue of (54), but are different types, yet with the property that their product (inherited from the matrix product) yields a scalar. A `point` (57) by itself has no algebraic properties, it only provides coordinates. However, the difference between two points is to be defined to yield a tangential vector (59). On `tvectors` and `covectors` usual matrix operations are inherently defined, so existing algorithms – that are usually provided using matrix algebra – can still be applied to them. However, objects that directly implement operations from Geometric Algebra such as `bivector`, `rotor` and `multivector` are safe from being used as parameters to matrix algebra, yet they inherit vector space properties. We can not show the actual implementation of the operations here due to space limitations; it is sufficient to emphasize that, by using C++ operator overloading, the API can be made very close to the mathematical notation. In addition it is convenient to overload the function call operator "`()`" for `rotor` objects to denote them to be applied to a vector object, meaning "$R(v)$" $:= RvR^{-1}$. This operator will be used in the following code excerpts.

4.2 Camera Navigation using GA

A "camera" in the Vish [1] visualization framework is defined through an observer's location P, a point that is looked at L, and an horizontal view plane, which is given as a bi-vector U corresponding to the "upwards" direction. The difference $t = L - P$ gives the view direction, a tangential vector.

One algorithm for camera navigation is to rotate the camera by an angle φ horizontally around the point of interest L and by an angle ϑ "upwards" along the line of sight. This algorithm is easily expressed in terms of geometric algebra. First we define the view plane V as

$$V := t \wedge \star U \tag{63}$$

and then construct a horizontal and a vertical rotor:

$$R_H := e^{U/|U|\ \varphi} \quad , \quad R_V := e^{\ V/|V|\ \vartheta} \quad . \tag{64}$$

Now the camera motion is achieved by computing the new observer location by adding the rotated view direction to the point of interest:

$$P_{new} = L + (R_H R_V)\,(t) \tag{65}$$

Finally, the horizontal view plane needs to be adjusted as well by the vertical rotation

$$U_{new} = R_V U \tag{66}$$

Another algorithm will rotate the camera around the view direction. This is trivial to implement, since we just need the rotor R_t that corresponds to the view direction, which is given by the exponential of the "sight vector's" dual ,

$$R_t = e^{\varphi\ \star(P-L)/|P-L|} \quad , \tag{67}$$

and apply this to the camera's Up-bivector to rotate it. This formulation is considered to be much simpler than an equivalent formulation using matrices and objects like "axial vectors". Using the operations and involved objects is very intuitive once their meaning in the Geometric Algebra has become clear.

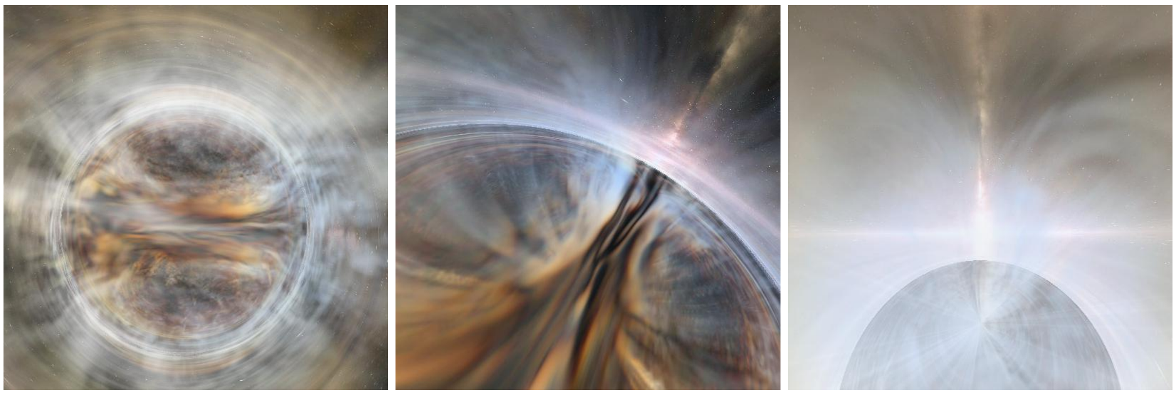


Figure 6: Three frames from a 3000-frame general relativistic volume-rendering with the BHFS of a general relativistic magnetohydrodynamic supercomputer simulation of a disk and jet around a black hole (John Hawley, 2007, private communication). The three frames show, from left to right, (a) outside the black hole, (b) passing through the black hole's outer horizon, (c) hitting the black hole's inner horizon, where the infinite blueshift and energy density triggers the mass inflation instability (Poisson & Israel 1990). The background texture was created from a 3D model of the Milky Way by Donna Cox's team at NCSA. The sequence was prepared for "Monster Black Hole", an episode of National Geographic's Naked Science series.

4.3 Relativistic observers in the BHFS

4.3.1 The BHFS

The Black Hole Flight Simulator (BHFS) is general relativistic software that can be used to visualize black holes. The BHFS remains work in progress, but has already been used in a number of productions, including the large-format high-resolution dome show "Black Holes: The Other Side of Infinity" (2006, Denver Museum of Nature and Science), and the TV documentaries "Monster of the Milky Way" (2006, NOVA-PBS), and "Monster Black Hole" (2008, Naked Science series, National Geographic). Figure 6 illustrates three frames from a sequence rendered for the National Geographic documentary.

The BHFS provides a complete implementation of the Reissner-Nordström geometry of a charged black hole, including its analytic connections inside the horizon to wormholes, white holes, and other universes. Real astronomical black holes probably have little charge, but they probably do rotate rapidly. A charged black hole is often taken as a surrogate for a rotating black hole, since the interior structure of a spherical charged black hole resembles that of a rotating black hole, but is much easier to model.

The Reissner-Nordström geometry, like its rotating counterpart the Kerr-Newman geometry, is subject to the relativistic counter-streaming instability at the inner horizon first pointed out by Poisson & Israel (1990) [10], and called by them "mass inflation" (see Hamilton & Avelino 2009 [5] for a review). The inflationary instability is expected to eliminate the wormhole and white hole connections inside realistic (astronomical) black holes.

4.3.2 Lorentz rotors in the BHFS

In addition to volume-rendering, the BHFS implements quasi-rigid objects, called "Ships", which by default move along geodesics in the black hole geometry. The camera (observer) is attached to one of the Ships. The orientation and motion of the camera are defined by a Lorentz transformation (which includes both a spatial rotation and a Lorentz boost), or equivalently, by a Lorentz rotor.

A Lorentz rotor R is a unimodular member of the even elements of the spacetime algebra. A Lorentz rotor can be written

$$R = e^{\theta} \tag{68}$$

where θ is a bivector in the spacetime algebra. The corresponding inverse Lorentz rotor is the reverse $\bar{R}$

$$\bar{R} = e^{-\theta} \ . \tag{69}$$

The condition of being unimodular means $\bar{R}R = 1$.

The even spacetime algebra is isomorphic to the algebra of complex quaternions, also called biquaternions. A complex quaternion can be written

$$q = q_R + I q_I \tag{70}$$

where q_R and q_I are two real quaternions comprising the real and imaginary parts of the complex quaternion q

$$q_R = ix_R + jy_R + kz_R + w_R \ , \quad q_I = ix_I + jy_I + kz_I + w_I \ . \tag{71}$$

The imaginary I is the pseudoscalar of the spacetime algebra. It commutes with the quaternionic imaginaries i, j, k. The quaternionic imaginaries themselves satisfy

$$i^2 = j^2 = k^2 = -1 \ , \quad ijk = 1 \ , \tag{72}$$

from which it follows that the quaternionic imaginaries anticommute between each other, for example $ij = -k = ji$. The convention $ijk = 1$, equation (72), agrees with the convention for quaternions in OpenGL, but is opposite to William Rowan Hamilton's carved-in-stone convention $ijk = -1$. In OpenGL, rotations accumulate to the right: a rotation $R = R_1 R_2$ means rotation R_1 followed by rotation R_2.

The BHFS stores a complex quaternion q as an 8-component object

$$q = \left(\begin{array}{cccc} x_R & y_R & z_R & w_R \\ x_I & y_I & z_I & w_I \end{array} \right) . \quad (73)$$

The reverse $\bar{q}$ of the complex quaternion q is its quaternionic conjugate

$$\bar{q} = \left(\begin{array}{cccc} -x_R & -y_R & -z_R & w_R \\ -x_I & -y_I & -z_I & w_I \end{array} \right) . \quad (74)$$

The group of Lorentz transformations, or Lorentz rotors, corresponds to complex quaternions of unit modulus. The unimodular condition $\bar{R}R = 1$, a complex condition, removes 2 degrees of freedom from the 8 degrees of freedom of complex quaternions, leaving the Lorentz group with 6 degrees of freedom, which is as it should be.

Spatial rotations correspond to real unimodular quaternions, and account for 3 of the 6 degrees of freedom of Lorentz transformations. A spatial rotation by angle θ right-handedly about the x-axis is the real Lorentz rotor

$$R = \cos(\theta/2) + i \sin(\theta/2) , \quad (75)$$

or, stored as a complex quaternion,

$$R = \left(\begin{array}{cccc} \sin(\theta/2) & 0 & 0 & \cos(\theta/2) \\ 0 & 0 & 0 & 0 \end{array} \right) . \quad (76)$$

Lorentz boosts account for the remaining 3 of the 6 degrees of freedom of Lorentz transformations. A Lorentz boost by velocity v, or equivalently by boost angle $\theta = \operatorname{atanh}(v)$, along the x-axis is the complex Lorentz rotor

$$R = \cosh(\theta/2) + Ii \sinh(\theta/2) , \quad (77)$$

or, stored as a complex quaternion,

$$R = \left(\begin{array}{cccc} 0 & 0 & 0 & \cosh(\theta/2) \\ \sinh(\theta/2) & 0 & 0 & 0 \end{array} \right) . \quad (78)$$

4.3.3 Simplicity of Lorentz rotors

The advantages of quaternions for implementing spatial rotations are well-known to 3D game programmers. Compared to standard rotation matrices, quaternions offer increased speed and require less storage, and their algebraic properties simplify interpolation and splining.

Complex quaternions retain similar advantages for implementing Lorentz transformations. They are fast, compact, and straightforward to interpolate or spline.

Under a spacetime rotation by Lorentz rotor R, a general multivector a in the spacetime algebra transforms as

$$a \rightarrow \bar{R} a R . \quad (79)$$

A general such multivector in the spacetime algebra is a 16-component object, with 8 even components, and 8 odd components.

As remarked earlier, the 8-component even spacetime subalgebra is isormorphic to the algebra of complex quaternions. As an example, the electromagnetic field constitutes a 6-component bivector, an even element of the spacetime algebra. The electric and magnetic fields E and B can be encoded as the complex quaternion

$$F = \left(\begin{array}{cccc} E_x & E_y & E_z & 0 \\ B_x & B_y & B_z & 0 \end{array} \right) . \quad (80)$$

The transformation (79) then becomes

$$F \rightarrow \bar{R} F R , \quad (81)$$

which is a powerful and elegant way to Lorentz transform the electromagnetic field. The electromagnetic field F in the transformation (81) is the complex quaternion (80), and the rotor R is another complex quaternion, so the Lorentz transformation (81) amounts to multiplying 3 complex quaternions.

The most common need in the BHFS is to Lorentz transform odd multivectors, not even multivectors. For example, every point on a scene that an observer sees is represented by the energy-momentum 4-vector of a photon emitted by the point and observed by the observer. Each such 4-vector $a = a^m \gamma_m$ is an odd multivector in the spacetime algebra. A general odd multivector is a sum of a vector part a and a pseudovector part Ib. The odd multivector can be written as a product of γ_t (the time basis element of the spacetime algebra) and an even multivector q

$$a + Ib = \gamma_t q \quad (82)$$

where q is the even multivector, or complex quaternion,

$$q = \left(\begin{array}{cccc} -b^x & -b^y & -b^z & a^t \\ a^x & a^y & a^z & b^t \end{array} \right) . \quad (83)$$

The Lorentz transformation (79) implies $\gamma_t q \rightarrow \bar{R} \gamma_t q R = \gamma_t \bar{R}^* q R$, where * denotes complex conjugation with respect to the peudoscalar imaginary I. It follows that the complex quaternion q, equation (83), transforms as

$$q \rightarrow \bar{R}^* q R . \quad (84)$$

The transformation (84) of the complex quaternion (83) provides a simple and elegant way to Lorentz transform a 4-vector a^m and 4-pseudovector Ib^m. Since b^m (without the I factor) is just another 4-vector, the transformation (84) effectively transforms two 4-vectors, a^m and b^m, simultaneously. The transformation (84) amounts to multiplying 3 complex quaternions.

5. VECTORS ON THE HARD DISK

5.1 Meta-Data on Vector Types

Storing a specific vector on hard disk, entails storing its numerical representation in a chosen coordinate system. However, when reading an unknown object from disk, information merely about its numerical representation is insufficient to determine the type vector (co-vector, bi-vector, ...). We need some meta-data, additional information about the data itself, that tells what properties the object on disk has.

C++ type trait templates proof useful here. Type traits are C++ templates that are specialized for known types to provide information on these types without the need to modify the type itself. An example of a type trait definition is given in the following code excerpt:

```
template <class Type> struct MetaInfo;

template <>      struct MetaInfo<double>
{   enum   { SIZE = 1 }   };

template <int N> struct MetaInfo<FixedArray<N, double> >
{   enum   { SIZE = N }   };
```

The type trait `MetaInfo` associates an integer value `SIZE` with an arbitrary type `Type`. This information is available at compile-time, and can be reduced to an usual integer in a template class at any time, such as in:

```
template <class Type> int NumberOfElements(const Type&T)
{   return MetaInfo<T>::SIZE;   }
```

Note that a type trait class may also specify default values (by specifying a non-specialized definition) and can be functions on template types itself (as demonstrated in the second specialization). This mechanism allows to equip existing types, e.g. as provided by external libraries, with meta-information as required for our framework.

The objective is to specify complete meta-information about a "vector space element" as required to uniquely identify it. As introduced in section 2, such information includes a reference to the metric (or metric field) and the orientation form ι. This information can be provided via a "coordinate system", which can be a global type definition – not more than providing the implicit knowledge on how to perform these operations, such as in Euclidean space. In such a case, no memory or computational resources are implied, but another type definition could require explicit formulae for expressions that are implicit in Euclidean space. Such a chart object may be expressed via a convention on how the coordinate functions are named, for instance $\{x, y, z\}$ for Cartesian coordinates versus $\{r, \vartheta, \varphi\}$ for polar coordinates. While this is yet work in progress, the following quantities have been found to be required for at least basic distinction and identification of vector types:

I *multiplicity*: an integer value expressing the number of components of this type.

II *rank*: the power $k = a + b$ of the vector space in terms of the tangential space $T^a(M) \times (T^*)^b(M)$; it is the dimensionality of the index space when considering the vector type as an array: zero indicates a scalar type, one is a one-dimensional vectorial type (tangential vector, co-vector, pseudo-vector, pseudo-covector), two are objects representable as matrix, etc.

III *grade*: for quantities from geometric algebra, specifies the grade k of the k-vector; the default is zero, for instance for symmetric tensor fields. For example, a bivector in 3D will have a grade of 2 whereas its rank is 1.

IV *dimensions*: the dimensionality n of the n-dimensional manifold on which this vector type is attached.

V *coordinatename(i)*: textual functions specifying the naming convention for each of the n coordinate functions.

VI *covariance(i)*: for each index, a flag specifying whether the index is an upper index or lower index. It can be implemented via some function that returns true or false for each index; this function may be evaluated fully at compile-time (a template function that is known) or via lookup into some static array.

VII *symmetries(n)*: often, tensors have symmetric or anti-symmetric index pairs. For efficiency reasons it is then important to calculate and store only a minimum subset of the components. This can be implemented via two lookup tables: one table lists those components which are actually stored, the other table contains the prescription for obtaining each tensor component. In a simple scheme, each tensor component is either stored, or is the negative of a stored component, or is zero. (See tables 1 and 2 for examples.) More complex schemes also allow cyclic symmetries, where tensor components can be linear combinations of stored components.

VIII *coordinate systems(i)*: tensor components are only defined with respect to a particular coordinate system. It is necessary to store (for each index) the name of the associated coordinate system. There are objects, such as basis systems or operators that transform between different coordinate systems, where different tensor indices correspond to different coordinate systems.

These properties have been chosen such that some operations on the given types can also succeed with partial knowledge, since certain algorithms do not require full knowledge of the entire algebraic operations of all types.

List of stored components mapping the component name to each storage index:

[0]	[1]	[2]	[3]	[4]	[5]
g_{xx}	g_{xy}	g_{xz}	g_{yy}	g_{yz}	g_{zz}

Obtaining tensor components from stored components via prescription for each entry:

g_{xx}	g_{xy}	g_{xz}	g_{yx}	g_{yy}	g_{yz}	g_{zx}	g_{zy}	g_{zz}
+[0]	+[1]	+[2]	+[1]	+[3]	+[4]	+[2]	+[4]	+[5]

Table 1: Storing a symmetric 3×3 tensor: The component table works like a pointer to the stored components.

List of stored components, mapping the component name to each storage index:

[0]	[1]	[2]
B_{xy}	B_{xz}	B_{yz}

Obtaining tensor components from stored components via prescription for each entry:

B_{xx}	B_{xy}	B_{xz}	B_{yx}	B_{yy}	B_{yz}	B_{zx}	B_{zy}	B_{zz}
0	+[0]	+[1]	−[0]	0	+[2]	−[1]	−[2]	0

Table 2: Storing an antisymmetric 3×3 tensor: The component table defines also signs during dereferencing, or in general, a polynomial expression of components.

This list of "vector properties" is not claimed to be complete; it is an early attempt to find a comprehensive scheme to cover all geometric and algebraic quantities that occur when performing numerical computations on manifolds. Special attention must also be given to the case of non-tensorial quantities such as Christoffel symbols, which do not yet fit into this ontology.

The Cactus framework [4, 13] currently uses a scheme that is simpler than the above; it is based on tensor algebra only and does not support *grades*. However, it does offer support for tensor densities (by associating a *weight* with each quantity), and it handles also certain special non-tensorial objects, such as logarithms of scalar densities and Christoffel symbols. These special cases are handled as exceptions; there is no generic scheme for them. This scheme is mostly used for symmetry conditions, which require either reflecting (mirroring) or rotating tensors. These operations require only the *symmetry* information above.

5.2 Storing Vector Types in HDF5

HDF5[12] is a generic scientific data format with supporting software, primarily an API provided in C. An HDF5 file can be viewed as a container, in which data objects are organized in ways that are meaningful and convenient to an application. HDF5 can be seen as a framework, rather than a specific format itself, allowing adaption to the various needs of diverse scientific domains [3]. The basic HDF5 object model is relatively simple, yet extremely versatile in terms of the types of data that it can store. The model contains two primary objects: groups, and datasets. Groups provide the organizing structures, and datasets are the basic storage structures. HDF5 groups and datasets may also have associated attributes, which are small data objects for storing metadata defined by applications.

HDF5 allows the specification of user-defined types that shall be stored in a file via its H5T API[2]. For instance, a struct in C/C++ of the form

```
struct CartesianVector { double x,y,z; };
```

can be expressed in the H5T API as *compound type*:

```
hid_t id = H5Tcreate(H5T_COMPOUND,
                     sizeof(CartesianVector) );
H5Tinsert( id, "x", 0, H5T_DOUBLE);
H5Tinsert( id, "y",   sizeof(double), H5T_DOUBLE);
H5Tinsert( id, "z", 2*sizeof(double), H5T_DOUBLE);
```

This code fragment creates an HDF5 identifier `id` that represents a type of the memory layout as in the aforementioned structure definition. This functionality provides an implementation of the component storage indices as used in table 1 and 2. More details can be found in the HDF5 reference manual.

When writing or reading a dataset to disk, the HDF5 API requires a type identifier to be specified with a `void*`. This tells the HDF5 library how to interpret some chunk of memory. Various generic tools exist to investigate the contents of an HDF5 file, which has a structure of a file system itself. "Datasets" play the role of a file, "Groups" the role of a directory. The tool `h5ls` – part of the HDF5 distribution – lists the contents of an HDF5 file in the fashion of the Unix tool `ls`, enhanced with additional information about the *type* of a dataset. The following example shows how a three-dimensional dataset `CartesianVector data[5][13][9];` appears in this file listing (shortened as compared with actual output):

```
/Block00001 Dataset {5/5, 13/13, 9/9}
    Location:  1:15768
    Links:     1
    Storage:   7020 allocated bytes
    Type:      struct {
                   "x"      +0    native float
                   "y"      +4    native float
                   "z"      +8    native float
               } 12 bytes
    Data:
      (0,0,0) {0.210951, -0.0406732, 0.0611351},
              {0.208286, -0.0525892, 0.0610958},
              ...
```

By virtue of HDF5, we can easily attach *names* to the purely numerical values in the data field. Hereby the HDF5 library offers various features that are very useful in practice, such as not only taking care of conversions between big-endian and little-endian platforms, but also conversions from double to float component types as well as transformations between different layouts such as $\{x, y, z\} \Leftrightarrow \{z, x, y\}$.

The availability of a naming scheme attached to numerical values is already sufficient to identify a coordinate system that is supposed to be "attached" to these numbers, in spirit of 5.1, V. Knowing the coordinate system relative to which the numbers are stored, in addition we need to specify the various attributes defining the algebraic properties of this vector type HDF5 allows to attach attributes with a dataset, group or "named data type". A named data type is a type id that was created by the `H5Tcreate()` call but saved to disk. It needs to be associated with a group in the file. Attributes attached to such a named data type are shared among all data sets of this type – the data type acts like a pointer to a common location of a set of attributes. We now need to define an HDF5 type for each of the vector types as defined from the meta-information about a specific data type. The following HDF5 listing shows the created named type, stored in a group `/Charts/Cartesian3D`, as it is named "Point" and equipped with an integer telling this data type refers to a manifold of dimension three. This data type "Point" is then later used to declare a dataset of points (shown with two attributes denoting the name of the associated chart and the dimension of the related manifold):

```
/Charts/Cartesian3D/Point Type
    Attribute: ChartDomain scalar
        Type:  null-terminated ASCII string
        Data:  "Cartesian3D"

    Attribute: Dimensions scalar
        Type:      native int
        Data:  3
    Type:      shared-1:13328 struct {
                   "x"     +0    native float
                   "y"     +4    native float
                   "z"     +8    native float
               } 12 bytes

/Block00001 Dataset {5/5, 13/13, 9/9}
    Location:  1:15768
    Links:     1
    Storage:   7020 allocated bytes
```

[2]`http://www.hdfgroup.org/HDF5/doc/RM/RM_H5T.html`

```
Type:      { shared-1:13328} struct {
               "x"      +0    native float
               "y"      +4    native float
               "z"      +8    native float
           } 12 bytes
Data:
```

This scheme allows to identify the dataset named "Blocks" as representing Cartesian coordinates of point locations. Accessing the dataset "Blocks" during reading, the software application can easily check for the attributes of the dataset to retrieve its algebraic properties. However, doing so is *optional*. Many applications might not implement the full set of tensor algebra, but might still provide a set of useful operations – such as displaying a dataset numerical as a spreadsheet etc. The information that a dataset consists of three floating point numbers, (the only information required for a generic operation such as displaying as spreadsheet) is available immediately. More complex properties require further lookup.

5.3 Storing Multi-Vector Types in HDF5

Multivectors are linear combinations of vectors of different basis elements, thereby forming an higher-dimensional space. A similar functionality is achieved using HDF5 by creating compound types from the basic vector types. For instance, given a bivector type in 3D, created by HDF5 API calls of the form

```
hid_t   bivector3D_id   =
       H5Tcreate( H5T_COMPOUND, 3*sizeof(double) );

H5Tinsert( bivector3D_id, "yz",  0, H5T_NATIVE_DOUBLE);
H5Tinsert( bivector3D_id, "zx",  8, H5T_NATIVE_DOUBLE);
H5Tinsert( bivector3D_id, "xy", 16, H5T_NATIVE_DOUBLE);
```

we may create a rotor in the following as compound containing the bivector, and adding a scalar:

```
hid_t   rotor3D_id   = H5Tcreate( H5T_COMPOUND, 32 );

H5Tinsert( rotor3D_id, "cos", 0, H5T_NATIVE_DOUBLE);
H5Tinsert( rotor3D_id, "sin", 8, bivector3D_id);
```

We name the scalar and bivector component "cos" and "sin" here, inspired by the construction of a rotor. What naming scheme to use here in general, will yet need to be explored. It is now a nice feature of HDF5 that different storage schemes are automatically mapped, i.e. datasets stored as the following type

```
hid_t antirotor3D_id =
H5Tcreate( H5T_COMPOUND, 4*sizeof(double) );
H5Tinsert( antirotor3D_id, "sin", 0 , bivector3D_id);
H5Tinsert( antirotor3D_id, "cos", 24, H5T_NATIVE_DOUBLE);
```

can be directly read without further specific treatment as a `rotor3D_id` dataset. This way HDF5 easily provides the notion of $a + c \wedge b \equiv c \wedge b + a$, i.e., commutativity of the "+" operator. One can also define a type which only retrieves the bivector component of a dataset of rotors, or the scalar component. This functionality is already provided by HDF5.

6. CONCLUSION

In this article we have reviewed the various types of what is usually called a "vector" in the context of differential geometry and geometric algebra. Various algebraic types have been identified, which are all represented numerically by three floating point numbers in three dimensions: tangential vectors, co-vectors, bi-vectors and bi-co-vectors. Yet these four different types have distinct algebraic properties and should be distinguished. We demonstrated the application of diverse vector types in four dimensions, leading to an easier formulation of the Newmann-Penrose formalism by virtue of Geometric Algebra. The clarity of the diverse algebraic types as achieved via GA thereby eases "navigation" in Riemann space, computer graphic applications (where two examples are given), and identification of quantities stored in files.

7. REFERENCES

[1] W. Benger, G. Ritter, and R. Heinzl. The Concepts of VISH. In 4th *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007.

[2] A. Bossavit. Differential geometry for the student of numerical methods in electromagnetism. Technical report, Tampere University of Technology, 1991. URL `http://butler.cc.tut.fi/~bossavit/`.

[3] M. T. Dougherty, M. J. Folk, E. Zadok, H. J. Bernstein, F. C. Bernstein, K. W. Eliceiri, W. Benger, and C. Best. Unifying biological image formats with hdf5. *Communications of the ACM (CACM)*, 52(10):42–47, October 2009.

[4] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[5] A. J. S. Hamilton and P. P. Avelino. The physics of the relativistic counter-streaming instability that drives mass inflation inside black holes. *Physics Reports*, accepted, 2009. gr-qc/0811.1926.

[6] A. Held. A formalism for the investigation of algebraically special metrics. i. *Commun. Math. Phys.*, 37:311–26, 1974.

[7] D. Hestenes. *New Foundations for Classical Mechanics, 2nd ed.* Springer Verlag., 1999.

[8] D. Hestenes. Oersted medal lecture 2002: Reforming the mathematical language of physics. *American Journal of Physics*, 71(2):104–121, 2003. URL `http://link.aip.org/link/?AJP/71/104/1`.

[9] E. T. Newman and R. Penrose. An approach to gravitational radiation by a method of spin coefficients. *J. Math. Phys.*, 3:566–79, 1962.

[10] E. Poisson and W. Israel. Inner-horizon instability and mass inflation in black holes. *Phys. Rev.*, D41:1796–1809, 1990.

[11] E. (Ted) Newman and R. Penrose. Spin-coefficient formalism. *Scholarpedia*, 4(6):7445, 2009.

[12] The HDF Group. Hierarchical data format version 5. `http://www.hdfgroup.org/HDF5`, 2000-2009.

[13] C. C. Toolkit. Cactus Computational Toolkit home page, URL `http://www.cactuscode.org/`.

Inertial Navigation using Geometric Algebra

Liam Candy
Department of Engineering
University of Cambridge, CB2 1PZ, UK
and CSIR, Pretoria, South Africa
lpc28@cam.ac.uk

Joan Lasenby
Department of Engineering
University of Cambridge, CB2 1PZ, UK
jl221@cam.ac.uk

Keywords: Inertial Navigation, Bortz Equation, Geometric Algebra, Conformal Algebra.

EXTENDED ABSTRACT

1 INTRODUCTION

In strapdown inertial navigation systems (SDINS) the angular velocity measured in a body frame is used to update a rotation generator that relates the orientation of the body frame to some reference frame. Because the angular velocity measurements are made in the body frame, which changes its orientation relative to the reference frame, direct integration of the conventional direction cosine differential equation is an unsuitable method for tracking attitude. Bortz [Bor71] derived a method of accounting for this effect by representing the actual finite rotation by an orientation vector $\phi(t)$ and obtaining an expression for $\dot{\phi}(t)$. Bortz showed that $\dot{\phi}$ could be split into two parts, $\omega(t)$ and $\dot{\sigma}(t)$. $\omega(t)$ is the angular rate vector and $\dot{\sigma}(t)$ is a term arising due to the change in the body coordinate system.

$\dot{\phi}$ in terms of ϕ and ω is known as the *Bortz Equation* – updating the quantity ϕ given the measurements and then extracting the attitude, is known to be more accurate than direct integration of the dynamical equations. In this paper we will use *geometric algebra (GA)* to show how the Bortz equation arises almost trivially from the dynamical equations.

Recent contributions in the SDINS literature have used dual quaternions, a concise representation for both rotations and translations, to extend the Bortz equation to the more general case of simultaneously updating both attitude and position [YWL05]. The claim is that there is an equivalent *Bortz* equation which allows more accurate updating. Here we use the *conformal geometric algebra* (CGA) [HS84] to investigate what a *conformal Bortz equation* would look like and whether the claims made for dual quaternions are valid.

The 3D and 5D Bortz equations

If the angular velocity of the body wrt the reference frame is written as a bivector in the body frame, Ω^b, and the rotor which takes the reference frame to the body frame at time t as $R(t)$, the dynamical equation is $\dot{R} = -\frac{1}{2}R\Omega^b$

We define Φ as $\Phi = \alpha B$, where B is a unit bivector and $R = \mathrm{e}^{-\Phi/2}$, ie a rotation of α in the plane of B. Since $\dot{\Phi} = \alpha\dot{B} + \dot{\alpha}B$, and using expressions for $\dot{\alpha}$ and $\dot{B}$ obtained by equating scalar and bivector parts of the dynamical equation, we are able to write $\dot{\Phi}$ as a bivector equation (where Ω^b is written as Ω for conciseness):

$$\dot{\Phi} = \Omega + \left(\frac{|\Phi|}{2}\cot\frac{|\Phi|}{2} - 1\right)\left[\Omega + \frac{(\Omega\cdot\Phi)\Phi}{|\Phi|^2}\right] - \frac{\langle\Phi\Omega\rangle_2}{2} \tag{1}$$

This is effectively the dual of the conventional Bortz equation.

Using the CGA framework we know that we can express both rotations and translations as rotors – thus we are able to define a new Φ in our 5D space as $\Phi = \alpha B + tn$, where αB gives the spatial rotation as before and t is related to the spatial translation. It is not hard to show that $R = \mathrm{e}^{-\Phi/2}$ represents a rotation of α in the plane of B followed by a translation of s which is a function of both t and the spatial rotation. We then also show that the equivalent 5D angular velocity bivector, Ω is given by $\Omega = \Omega_3 + n\tilde{R}_\alpha \dot{s} R_\alpha$, where Ω_3 is the 3D angular velocity bivector.

The 5D dynamical equation is again $\dot{R} = -\frac{1}{2}R\Omega$. Equating scalar and spatial bivector parts gives expressions for $\dot{\alpha}$ and $\dot{B}$ (indeed those obtained in the 3D case). Equating non-spatial bivector and 4-vector parts simply gives identities, thus confirming consistency.

We then form $\dot{\Phi} = \dot{\alpha}B + \alpha\dot{B} + \dot{t}n$ in terms of Φ and Ω. However, the resulting equation is not as simple in form as the 3D equation, ie it is not simply a matter of replacing ϕ and ω with their 5D counterparts. This therefore calls into question the process of substituting a dual quaternion generator into the Bortz equation form and updating the dual quaternion components. The paper will discuss the differences and investigate the discrepancies via simulations.

REFERENCES

[Bor71] J.E Bortz. A new mathematical formulation for strapdown inertial navigation. *IEEE Transactions on Aerospace and Electronic Systems*, AES-7, no. 1:61–66, 1971.

[HS84] D Hestenes and G Sobczyk. *Clifford Algebra to Geometric Calculus: A unified language for mathematics and physics*. 1984.

[YWL05] D. Hu T. Li Y. Wu, X. Hu and J. Lian. Strapdown inertial navigation system algorithms based on dual quaternions. *IEEE Transactions on Aerospace and Electronic Systems*, 41-4:110–132, 2005.

Clifford (Geometric) Algebra Wavelet Transform

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

January 22, 2010

Abstract

While the Clifford (geometric) algebra Fourier Transform (CFT) is global, we introduce here the local Clifford (geometric) algebra (GA) wavelet concept. We show how for $n = 2, 3 (\text{mod } 4)$ continuous Cl_n-valued admissible wavelets can be constructed using the similitude group $SIM(n)$. We strictly aim for real geometric interpretation, and replace the imaginary unit $i \in \mathbb{C}$ therefore with a GA blade squaring to -1. Consequences due to non-commutativity arise. We express the admissibility condition in terms of a Cl_n CFT and then derive a set of important properties such as dilation, translation and rotation covariance, a reproducing kernel, and show how to invert the Clifford wavelet transform. As an explicit example, we introduce Clifford Gabor wavelets. We further invent a generalized Clifford wavelet uncertainty principle. Extensions of CFTs and Clifford wavelets to $Cl_{0,n'}, n' = 1, 2 (\text{mod } 4)$ appear straight forward.

Keywords: Clifford geometric algebra, Clifford wavelet transform, multidimensional wavelets, continuous wavelets, similitude group.
AMS Subj. Class.: 15A66, 42C40, 94A12.

1 Introduction

The meaning and importance of wavelets is clearly seen in a biographical note on J. P. Morlet: *Following in the footsteps of Denis Gabor (father of holography), Morlet was disconcerted by the poor results he [Gabor] obtained; but, being inquisitive and persistent, he asked himself, "Why?" and immediately provided the answer. Gabor paved the time-frequency plane in uniform cells and associated each cell with a wave shape of invariant envelope with a carrier of variable frequency. Morlet kept the constraint resulting from the uncertainty principle applied to time and frequency, but he perceived that it was the wave shape that must be invariant to give uniform resolution in the entire plane. For this he adapted the sampling rate to the frequency, thereby creating, in effect, a changing time scale producing a stretching of the wave shape. Today the wavelet transform is also called the "time-scale analysis" approach, which is comparable to the conventional time-frequency analysis. . . . It has been rediscovered as a very useful tool, particularly in data compression where it can produce significant savings in storage and transmission costs but also in mathematics, data processing, communications, image analysis, and many other engineering problems.*[1]

In order to favorably combine wavelet techniques with Clifford (geometric) algebra, which provides a complete algebra of a vector space and all its subspaces, several efforts have been undertaken. They include Clifford multi resolution analysis (MRA) [2], quaternion MRA [4], Clifford wavelet networks, quaternion wavelet transforms (QWT) applied to image analysis (using the QWT phase concept), image processing and motion estimation [5], quaternion-valued admissible wavelets, Clifford algebra-valued admissible (continuous) wavelets using complex Fourier transforms for the spectral representation [6], monogenic wavelets over the unit ball [7], Clifford continuous wavelet transforms (ContWT) in $L_{0,2}$, $L_{0,3}$, wavelets on the 3D sphere with Cauchy kernel in Clifford analysis (2009), diffusion wavelets [8], ContWT in Clifford analysis, wavelet frames on the sphere, benchmarking of 3D Clifford wavelet functions, metric dependent Clifford analysis, new multivariable polynomials and associated ContWT: Clifford versions of Hermite, Hermitean Clifford-Hermite, bi-axial Clifford-Hermite, Jacobi, Gegenbauer, Laguerre, and Bessel polynomials [3].

Fourier transformations have been successfully developed in the framework of real Clifford (geometric) algebra (GA), replacing the imaginary unit $i \in \mathbb{C}$ by a geometric (GA) square root of -1 [9]. These Clifford Fourier transformations (CFT) [10–12] have already found interesting applications in vector field analysis and pattern matching [17]. A special case are the socalled quaternion Fourier transforms (QFT) [13–15].

We now use the spectral CFT representation in order to develop real Clifford GA wavelets in dimensions $n = 2, 3 (\text{mod } 4)$. We dimensionally extend [16] and elaborate the short summary given in [18] by adding proofs and generalizations.

In Section 2 Clifford (geometric) algebra is introduced including multivector signal functions, the Clifford Fourier transform, and the similitude group of dilations, rotations and translations. Section 3 defines Clifford mother and daughter wavelets, spectral representation, discusses admissibility, the Clifford wavelet transformation and its spectral CFT representation. This is followed by a detailed discussion of Clifford wavelet properties, i.e. linearity, covariance w.r.t. dilation, rotation and translation, inner product and norm relations, the inverse Clifford wavelet transform, a reproducing kernel and a Clifford wavelet uncertainty principle. Finally the example of Clifford Gabor wavelets is given.

2 Clifford (geometric) algebra and multivector signals

2.1 Clifford (geometric) algebra

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

$$\boldsymbol{a}\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}, \tag{1}$$

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 $\boldsymbol{a}, \boldsymbol{b}$.

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

$$\{1, \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{\text{vectors}}, \underbrace{\mathbf{e}_2\mathbf{e}_3, \mathbf{e}_3\mathbf{e}_1, \mathbf{e}_1\mathbf{e}_2}_{\text{area bivectors}}, \underbrace{i = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3}_{\text{volume trivector}}\}. \tag{2}$$

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

$$\{\boldsymbol{e}_{k_1}\boldsymbol{e}_{k_2} \ldots \boldsymbol{e}_{k_r} \mid 1 \leq k_1 < k_2 < \ldots < k_r \leq n\}, \tag{3}$$

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 ($0 \leq k \leq n$): 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 $\langle M \rangle_n \in \bigwedge^n \mathbb{R}^{p,q}$

$$M = \sum_{A=1}^{2^n} M_A \boldsymbol{e}_A = \langle M \rangle + \langle M \rangle_1 + \langle M \rangle_2 + \ldots + \langle M \rangle_n \,. \tag{4}$$

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

$$\widetilde{M} = \sum_{k=0}^{n} (-1)^{\frac{k(k-1)}{2}} \langle M \rangle_k, \tag{5}$$

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

$$M * \widetilde{N} = \langle M\widetilde{N} \rangle = \langle M\widetilde{N} \rangle_0. \tag{6}$$

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

$$|M|^2 = M * \widetilde{M} = \sum_A M_A^2. \tag{7}$$

For $n = 2(\text{mod } 4)$ and $n = 3(\text{mod } 4)$ the pseudoscalar is $i_n = \boldsymbol{e}_1\boldsymbol{e}_2 \dots \boldsymbol{e}_n$ with (also valid for[1] $Cl_{0,n'}$, $n' = 1, 2(\text{mod } 4)$)

$$i_n^2 = -1. \tag{8}$$

A blade $B_k = \boldsymbol{b}_1 \wedge \boldsymbol{b}_2 \wedge \dots \wedge \boldsymbol{b}_k, \boldsymbol{b}_l \in \mathbb{R}^{p,q}, 1 \le l \le k \le n = p + q$ describes a k-dimensional vector *subspace*

$$V_B = \{\boldsymbol{x} \in \mathbb{R}^{p,q} | \boldsymbol{x} \wedge B = 0\}. \tag{9}$$

Its dual blade

$$B^* = B i_n^{-1} \tag{10}$$

describes the *complimentary* $(n - k)$-dimensional vector subspace $V_B^{\perp}$. The pseudoscalar $i_n \in Cl_n$ is *central* for $n = 3(\text{mod } 4)$

$$i_n\, M = M\, i_n, \qquad \forall M \in Cl_n. \tag{11}$$

But for even n we get due to non-commutativity [11] of the pseudoscalar $i_n \in Cl_n$ for all $M \in Cl_n, \lambda \in \mathbb{R}$

$$i_n M = M_{even}\, i_n - M_{odd}\, i_n\,, \tag{12}$$

$$e^{i_n \lambda} M = M_{even}\, e^{i_n \lambda} + M_{odd}\, e^{-i_n \lambda}. \tag{13}$$

2.2 Multivector signal functions

A multivector valued function $f : \mathbb{R}^{p,q} \to Cl_{p,q},\ p + q = n$, has 2^n blade components $(f_A : \mathbb{R}^{p,q} \to \mathbb{R})$

$$f(\boldsymbol{x}) = \sum_A f_A(\boldsymbol{x}) \boldsymbol{e}_A. \tag{14}$$

We define the *inner product* of $\mathbb{R}^n \to Cl_n$ functions f, g by

$$(f, g) = \int_{\mathbb{R}^n} f(\boldsymbol{x}) \widetilde{g(\boldsymbol{x})}\; d^n\boldsymbol{x} \tag{15}$$

$$= \sum_{A,B} \boldsymbol{e}_A \widetilde{\boldsymbol{e}_B} \int_{\mathbb{R}^n} f_A(\boldsymbol{x}) g_B(\boldsymbol{x})\; d^n\boldsymbol{x}, \tag{16}$$

and the $L^2(\mathbb{R}^n; Cl_n)$-*norm*

$$\|f\|^2 = \langle (f, f) \rangle = \int_{\mathbb{R}^n} |f(\boldsymbol{x})|^2 d^n\boldsymbol{x}$$

$$= \sum_A \int_{\mathbb{R}^n} f_A^2(\boldsymbol{x})\; d^n\boldsymbol{x}, \tag{17}$$

$$L^2(\mathbb{R}^n; Cl_n) = \{f : \mathbb{R}^n \to Cl_n \mid \|f\| < \infty\}. \tag{18}$$

[1]For an extension of the current real Clifford (geometric) algebra wavelet approach to Clifford algebras $Cl_{0,n'}$ the definition of reversion has to be modified to include sign changes of negative definite basis vectors $\boldsymbol{e}_k \to \widetilde{\boldsymbol{e}}_k = -\boldsymbol{e}_k, 1 \le k \le n'$.

For the *Clifford geometric algebra Fourier transformation* (CFT) [11] the complex unit $i \in \mathbb{C}$ is replaced by some *geometric* (square) *root* of -1, e.g. pseudoscalars i_n, $n = 2, 3(\text{mod } 4)$. Complex functions f are replaced by multivector functions $f \in L^2(\mathbb{R}^n; Cl_n)$.

Definition 1 (Clifford geometric algebra Fourier transformation (CFT))**.** *The Clifford GA Fourier transform*[2] $\mathcal{F}\{f\}$*:* $\mathbb{R}^n \to Cl_n$, $n = 2, 3(\text{mod}\, 4)$ *is given by*

$$\mathcal{F}\{f\}(\boldsymbol{\omega}) = \widehat{f}(\boldsymbol{\omega}) = \int_{\mathbb{R}^n} f(\boldsymbol{x})\, e^{-i_n \boldsymbol{\omega}\cdot\boldsymbol{x}} d^n\boldsymbol{x}, \tag{19}$$

for multivector functions f*:* $\mathbb{R}^n \to Cl_n$*.*

The CFT (19) is *inverted* by

$$f(\boldsymbol{x}) = \mathcal{F}^{-1}[\mathcal{F}\{f\}(\boldsymbol{\omega})]$$

$$= \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} \mathcal{F}\{f\}(\boldsymbol{\omega})\, e^{i_n \boldsymbol{\omega}\cdot\boldsymbol{x}}\; d^n\boldsymbol{\omega}. \tag{20}$$

The *similitude group* $\mathcal{G} = SIM(n)$ of dilations, rotations and translations is a subgroup of the affine group of $\mathbb{R}^n$

$$\mathcal{G} = \mathbb{R}^+ \times SO(n) \rtimes \mathbb{R}^n$$

$$= \{(a, r_{\boldsymbol{\theta}}, \boldsymbol{b}) | a \in \mathbb{R}^+, r_{\boldsymbol{\theta}} \in SO(n), \boldsymbol{b} \in \mathbb{R}^n\}. \tag{21}$$

The *left Haar measure* on $\mathcal{G}$ is given by

$$d\lambda = d\lambda(a, \boldsymbol{\theta}, \boldsymbol{b}) = d\mu(a, \boldsymbol{\theta}) d^n\boldsymbol{b}, \tag{22}$$

$$d\mu = d\mu(a, \boldsymbol{\theta}) = \frac{da d\boldsymbol{\theta}}{a^{n+1}}, \tag{23}$$

where $d\boldsymbol{\theta}$ is the Haar measure on $SO(n)$. For example

$$d\boldsymbol{\theta} = \begin{cases} \frac{d\theta}{2\pi}, & n = 2 \\ \frac{1}{8\pi^2} \sin\theta_1 d\theta_1 d\theta_2 d\theta_3, & n = 3 \end{cases}. \tag{24}$$

We define the *inner product* of $f, g : \mathcal{G} \to Cl_n$ by

$$(f, g) = \int_{\mathcal{G}} f(a, \boldsymbol{\theta}, \boldsymbol{b}) \widetilde{g(a, \boldsymbol{\theta}, \boldsymbol{b})}\; d\lambda(a, \boldsymbol{\theta}, \boldsymbol{b}), \tag{25}$$

and the $L^2(\mathcal{G}; Cl_n)$-*norm*

$$\|f\|^2 = \langle (f, f) \rangle = \int_{\mathcal{G}} |f(a, \boldsymbol{\theta}, \boldsymbol{b})|^2 d\lambda, \tag{26}$$

$$L^2(\mathcal{G}; Cl_n) = \{f : \mathcal{G} \to Cl_n \mid \|f\| < \infty\}. \tag{27}$$

[2]The CFT can be defined analogously for $Cl_{0,n'}$, $n' = 1, 2(\text{mod } 4)$.

The variations of a multivector signal $f \in L^2(\mathbb{R}^n; Cl_n)$ in position $\boldsymbol{x} \in \mathbb{R}^n$ and frequency $\boldsymbol{\omega} \in \mathbb{R}^n$ are related by the *CFT uncertainty principle* [11, 19–21]

$$\|\boldsymbol{x}f\|^2_{L^2(\mathbb{R}^n;Cl_n)} \, \|\boldsymbol{\omega}\hat{f}\,\|^2_{L^2(\mathbb{R}^n;Cl_n)} \geq n\frac{(2\pi)^n}{4}\|f\|^4_{L^2(\mathbb{R}^n;Cl_n)}. \qquad (28)$$

3 Clifford GA wavelets

3.1 Real admissible continuous Clifford GA wavelets

We represent the transformation group $\mathcal{G} = SIM(n)$ by applying translations, scaling and rotations to a so-called *Clifford mother wavelet* $\psi : \mathbb{R}^n \to Cl_n$

$$\psi(\boldsymbol{x}) \longmapsto \psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x}) = \frac{1}{a^{n/2}}\psi(r_{\boldsymbol{\theta}}^{-1}(\frac{\boldsymbol{x}-\boldsymbol{b}}{a})). \qquad (29)$$

The family of wavelets $\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}$ are so-called *Clifford daughter wavelets.*

Lemma 1 (Norm identity)**.** *The factor $a^{-n/2}$ in $\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}$ ensures (independent of $a, \boldsymbol{\theta}, \boldsymbol{b}$) that*

$$\|\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}\|_{L^2(\mathbb{R}^n;Cl_n)} = \|\psi\|_{L^2(\mathbb{R}^n;Cl_n)}. \qquad (30)$$

Proof.

$$\begin{aligned}
&\|\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}\|^2_{L^2(\mathbb{R}^n;Cl_n)} \\
&= \int_{\mathbb{R}^n}\sum_A \frac{1}{a^n}\psi_A^2(\underbrace{r_{\boldsymbol{\theta}}{}^{-1}(\frac{\boldsymbol{x}-\boldsymbol{b}}{a})}_{=\boldsymbol{z}})\,d^n\boldsymbol{x} \\
&= \frac{1}{a^n}\int_{\mathbb{R}^n}\sum_A \psi_A^2(\boldsymbol{z})a^n\det(r_{\boldsymbol{\theta}})\,d^n\boldsymbol{z} \\
&= \int_{\mathbb{R}^n}\sum_A \psi_A^2(\boldsymbol{z})\,d^n\boldsymbol{z} = \|\psi\|_{L^2(\mathbb{R}^n;Cl_{n,0})}. \qquad (31)
\end{aligned}$$

□

The *spectral* CFT representation of Clifford daughter wavelets is

$$\mathcal{F}\{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}\}(\boldsymbol{\omega}) = a^{\frac{n}{2}}\widehat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))e^{-i_n\boldsymbol{b}\cdot\boldsymbol{\omega}}. \qquad (32)$$

In the proof of (32) the CFT properties of scaling, $\boldsymbol{x}$-shift and rotation are applied. A Clifford mother wavelet $\psi \in L^2(\mathbb{R}^n; Cl_n)$ is *admissible* if

$$\begin{aligned}
C_\psi &= \int_{\mathbb{R}^+}\int_{S0(n)} a^n\{\widehat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))\}^\sim\widehat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))\,d\mu \\
&= \int_{\mathbb{R}^n}\frac{\widetilde{\widehat{\psi}}(\boldsymbol{\omega})\widehat{\psi}(\boldsymbol{\omega})}{|\boldsymbol{\omega}|^n}\,d^n\boldsymbol{\omega}, \qquad (33)
\end{aligned}$$

is an *invertible multivector constant and finite* at a.e. $\boldsymbol{\omega} \in \mathbb{R}^n$. We must therefore have $\widehat{\psi}(\boldsymbol{\omega} = 0) = 0$

$$\begin{aligned}
\hat{\psi}(0) &= \int_{\mathbb{R}^n}\psi(\boldsymbol{x})e^{i_n 0\cdot\boldsymbol{x}}\,d^n\boldsymbol{x} = \int_{\mathbb{R}^n}\psi(\boldsymbol{x})\,d^n\boldsymbol{x} \\
&= \sum_A\int_{\mathbb{R}^n}\psi_A(\boldsymbol{x})\,d^n\boldsymbol{x}\,\boldsymbol{e}_A = 0, \qquad (34)
\end{aligned}$$

and therefore for *all* 2^n Clifford mother wavelet components

$$\int_{\mathbb{R}^n}\psi_A(\boldsymbol{x})\,d^n\boldsymbol{x} = 0. \qquad (35)$$

By construction $C_\psi = \widetilde{C_\psi}$. Hence for $n = 2, 3 (\mathrm{mod}\ 4)$

$$\begin{aligned}
C_\psi &= \langle C_\psi\rangle_0 + \langle C_\psi\rangle_1 + \langle C_\psi\rangle_4 + \langle C_\psi\rangle_5 + \ldots \\
&= \sum_{k=0}^{[n/4]}(\langle C_\psi\rangle_{4k} + \langle C_\psi\rangle_{4k+1}), \qquad (36)
\end{aligned}$$

and

$$\begin{aligned}
\langle C_\psi\rangle_0 &= \int_{\mathbb{R}^n}\langle\{\widehat{\psi}(\boldsymbol{\xi})\}^\sim\widehat{\psi}(\boldsymbol{\xi})\rangle_0\,\frac{1}{|\boldsymbol{\xi}|^n}d\boldsymbol{\xi}^n \\
&= \int_{\mathbb{R}^n}\frac{|\widehat{\psi}(\boldsymbol{\xi})|^2}{|\boldsymbol{\xi}|^n}d\boldsymbol{\xi}^n > 0. \qquad (37)
\end{aligned}$$

The invertibility of C_ψ depends on its grade content, e.g. for $n = 2, 3$, C_ψ is invertible, if and only if $\langle C_\psi\rangle_1^2 \neq \langle C_\psi\rangle_0^2$:

$$C_\psi^{-1} = \frac{\langle C_\psi\rangle_0 - \langle C_\psi\rangle_1}{\langle C_\psi\rangle_0^2 - \langle C_\psi\rangle_1^2}. \qquad (38)$$

Definition 2 (Clifford GA wavelet transformation (CWT))**.** *For an admissible GA mother wavelet $\psi \in L^2(\mathbb{R}^n; Cl_n)$ and a multivector signal function $f \in L^2(\mathbb{R}^n; Cl_n)$*

$$T_\psi : L^2(\mathbb{R}^n; Cl_n) \to L^2(\mathcal{G}; Cl_n), \qquad (39)$$

$$f \mapsto T_\psi f(a, \boldsymbol{\theta}, \boldsymbol{b}) = \int_{\mathbb{R}^n} f(\boldsymbol{x})\widetilde{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})}\,d^n\boldsymbol{x}. \qquad (40)$$

- Because of (12) we need to *restrict* the mother wavelet ψ for $n = 2 (\mathrm{mod}\ 4)$ to even or odd grades: Either we have a *spinor wavelet* $\psi \in L^2(\mathbb{R}^n; Cl_n^+)$ with $\varepsilon = 1$, or we have an *odd parity wavelet* $\psi \in L^2(\mathbb{R}^n; Cl_n^-)$ with $\varepsilon = -1$.

- For $n = 3 (\mathrm{mod}\ 4)$, no grade restrictions exist. We then always have $\varepsilon = 1$.

NB: The admissibility constant C_ψ is always *scalar* for $n = 2$, for the spinor wavelet as well as for the odd parity vector wavelet.

The *spectral* (CFT) representation of the Clifford wavelet transform is

$$\begin{aligned} &T_\psi f(a, \boldsymbol{\theta}, \boldsymbol{b}) \qquad (41)\\ &= \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \widehat{f}(\boldsymbol{\omega})\, a^{\frac{n}{2}} \{\widehat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))\}^{\sim} e^{\varepsilon i_n \boldsymbol{b}\cdot\boldsymbol{\omega}}\, d^n\boldsymbol{\omega}. \end{aligned}$$

Proof.

$$\begin{aligned} T_\psi f(a, \boldsymbol{\theta}, \boldsymbol{b}) &\overset{IP}{=} (f, \psi_{a,\boldsymbol{\theta},\boldsymbol{b}})_{L^2(\mathbb{R}^n;Cl_{n,0})} \qquad (42)\\ &\overset{PT}{=} \frac{1}{(2\pi)^n}(\widehat{f}, \widehat{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}})_{L^2(\mathbb{R}^n;Cl_{n,0})}\\ &= \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\omega}) \left[\widehat{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}}(\boldsymbol{\omega})\right]^{\sim} d^n\boldsymbol{\omega}\\ &\overset{FT}{=} \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\omega})\, e^{i_n \boldsymbol{b}\cdot\boldsymbol{\omega}} a^{\frac{n}{2}} \left[\widehat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))\right]^{\sim} d^n\boldsymbol{\omega}\\ &= \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\omega})\, a^{\frac{n}{2}} \left[\widehat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))\right]^{\sim} e^{\varepsilon i_n \boldsymbol{b}\cdot\boldsymbol{\omega}} d^n\boldsymbol{\omega}, \end{aligned}$$

with IP = inner product (15), PT = CFT Plancherel theorem [11], and $FT = \mathcal{F}\{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}\}$ of (32). □

NB: The CFT for $n = 2 (\mathrm{mod}\ 4)$ *preserves* even and odd grades.

3.2 Properties of real Clifford GA wavelets

We immediately see from Definition 2 that the Clifford GA wavelet transform is *left linear* with respect to multivector constants $\lambda_1, \lambda_2 \in Cl_n$.

We further have the following set of properties. *Translation covariance*: If the argument of $T_\psi f(\boldsymbol{x})$ is translated by a constant $\boldsymbol{x}_0 \in \mathbb{R}^n$ then

$$[T_\psi f(\cdot - \boldsymbol{x}_0)](a, \boldsymbol{\theta}, \boldsymbol{b}) = T_\psi f(a, \boldsymbol{\theta}, \boldsymbol{b} - \boldsymbol{x}_0)\,. \qquad (43)$$

Proof. By definition

$$\begin{aligned} &[T_\psi f(\cdot - \boldsymbol{x}_0)](a, \boldsymbol{\theta}, \boldsymbol{b})\\ &= \int_{\mathbb{R}^n} f(\boldsymbol{x} - \boldsymbol{x}_0)\widetilde{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}}(\boldsymbol{x})\, d^n\boldsymbol{x}\\ &= \int_{\mathbb{R}^n} f(\underbrace{\boldsymbol{x} - \boldsymbol{x}_0}_{=\boldsymbol{y}})\frac{1}{a^{\frac{n}{2}}} \left[\psi(r_{\boldsymbol{\theta}}^{-1}(\frac{\boldsymbol{x} - \boldsymbol{b}}{a})\right]^{\sim} d^n\boldsymbol{x}\\ &= \int_{\mathbb{R}^n} f(\boldsymbol{y})\frac{1}{a^{\frac{n}{2}}} \left[\psi\left(r_{\boldsymbol{\theta}}^{-1}(\frac{\boldsymbol{y} - (\boldsymbol{b} - \boldsymbol{x}_0)}{a})\right)\right]^{\sim} d^n\boldsymbol{y}\\ &= T_\psi f(a, \boldsymbol{\theta}, \boldsymbol{b} - \boldsymbol{x}_0). \qquad (44) \end{aligned}$$

□

Dilation covariance: If $0 < c \in \mathbb{R}$ then

$$[T_\psi f(c\,\cdot)](a, \boldsymbol{\theta}, \boldsymbol{b}) = \frac{1}{c^{\frac{n}{2}}} T_\psi f(ca, \boldsymbol{\theta}, c\boldsymbol{b})\,. \qquad (45)$$

Proof. By definition

$$\begin{aligned} &[T_\psi f(c\,\cdot)](a, \boldsymbol{\theta}, \boldsymbol{b})\\ &= \int_{\mathbb{R}^n} f(c\boldsymbol{x})\frac{1}{a^{\frac{n}{2}}} \left[\psi(r_{\boldsymbol{\theta}}^{-1}(\frac{\boldsymbol{x} - b}{a}))\right]^{\sim} d^n\boldsymbol{x}\\ &\overset{\boldsymbol{y} = c\boldsymbol{x}}{=} \int_{\mathbb{R}^n} f(\boldsymbol{y})\frac{1}{a^{\frac{n}{2}}}\frac{1}{c^n} \left[\psi\left(r_{\boldsymbol{\theta}}^{-1}(\frac{\frac{\boldsymbol{y}}{c} - \boldsymbol{b}}{a})\right)\right]^{\sim} d^n\boldsymbol{y}\\ &= \frac{1}{c^{\frac{n}{2}}}\int_{\mathbb{R}^n} f(\boldsymbol{y})\frac{1}{(ac)^{\frac{n}{2}}} \left[\psi\left(r_{\boldsymbol{\theta}}^{-1}(\frac{\boldsymbol{y} - \boldsymbol{b}c}{ac})\right)\right]^{\sim} d^n\boldsymbol{y}\\ &= \frac{1}{c^{\frac{n}{2}}} T_\psi f(ac, \boldsymbol{\theta}, \boldsymbol{b}c). \qquad (46) \end{aligned}$$

□

Rotation covariance: If $r = r_{\boldsymbol{\theta}}$, $r_0 = r_{\boldsymbol{\theta_0}}$ and $r' = r_{\boldsymbol{\theta}'} = r_0 r = r_{\boldsymbol{\theta_0}} r_{\boldsymbol{\theta}}$ are rotations, then

$$[T_\psi f(r_{\boldsymbol{\theta_0}}\cdot)](a, \boldsymbol{\theta}, \boldsymbol{b}) = T_\psi f(a, \boldsymbol{\theta}', r_{\boldsymbol{\theta_0}}\boldsymbol{b})\,. \qquad (47)$$

Proof. By definition and with substitution $\boldsymbol{y} = r_0\boldsymbol{x}$

$$\begin{aligned} &[T_\psi f(r_0\cdot)](a, \boldsymbol{\theta}, \boldsymbol{b})\\ &= \int_{\mathbb{R}^n} f(r_0\boldsymbol{x})\widetilde{\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}}(\boldsymbol{x})\, d^n\boldsymbol{x}\\ &= \int_{\mathbb{R}^n} f(r_0\boldsymbol{x}) \left[\psi(r^{-1}(\frac{\boldsymbol{x} - \boldsymbol{b}}{a}))\right]^{\sim} d^n\boldsymbol{x}\\ &= \int_{\mathbb{R}^n} f(\boldsymbol{y}) \left[\psi\left(r^{-1}(\frac{r_0^{-1}\boldsymbol{y} - \boldsymbol{b}}{a})\right)\right]^{\sim} \det{}^{-1}(r)\, d^n\boldsymbol{y}\\ &= \int_{\mathbb{R}^n} f(\boldsymbol{y}) \left[\psi\left(r^{-1}r_0^{-1}(\frac{\boldsymbol{y} - r_0\boldsymbol{b}}{a})\right)\right]^{\sim} d^n\boldsymbol{y}\\ &= \int_{\mathbb{R}^n} f(\boldsymbol{y}) \left[\psi\left((r_0 r)^{-1}(\frac{\boldsymbol{y} - r_0\boldsymbol{b}}{a})\right)\right]^{\sim} d^n\boldsymbol{y}\\ &= T_\psi f(a, \boldsymbol{\theta}', r_0\boldsymbol{b}). \qquad (48) \end{aligned}$$

□

Now we will see some *differences* from the classical wavelet transforms.

The next property is an *inner product relation*: Let $f, g \in L^2(\mathbb{R}^n; Cl_n)$ arbitrary. Then we have

$$(T_\psi f, T_\psi g)_{L^2(\mathcal{G};Cl_n)} = (fC_\psi, g)_{L^2(\mathbb{R}^n;Cl_n)} \qquad (49)$$

In the following proof of (49) we will use the abbreviations

$$F_{a,\boldsymbol{\theta}}(\boldsymbol{\omega}) = a^{\frac{n}{2}} \hat{f}(\boldsymbol{\omega})\{\hat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}))\}^{\sim}, \qquad (50)$$

$$G_{a,\boldsymbol{\theta}}(\boldsymbol{\omega}') = a^{\frac{n}{2}} \hat{g}(\boldsymbol{\omega}')\{\hat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\omega}'))\}^{\sim}\,, \qquad (51)$$

and the spectral representations (41)

$$T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b}) = \frac{\mathcal{F}\{F_{a,\boldsymbol{\theta}}\}(-\varepsilon\boldsymbol{b})}{(2\pi)^n}, \tag{52}$$

$$T_\psi g(a,\boldsymbol{\theta},\boldsymbol{b}) = \frac{\mathcal{F}\{G_{a,\boldsymbol{\theta}}\}(-\varepsilon\boldsymbol{b})}{(2\pi)^n}. \tag{53}$$

Proof. Using the abbreviations (50), (51) and spectral representations (52), (53) we get

$$\begin{aligned}
&(T_\psi f, T_\psi g)_{L^2(\mathcal{G};Cl_{3,0})} \\
&= \frac{1}{(2\pi)^{2n}} \int_{\mathbb{R}^+}\int_{S0(n)} \bigg(\int_{\mathbb{R}^n} \mathcal{F}\{F_{a,\boldsymbol{\theta}}\}(-\varepsilon\boldsymbol{b}) \\
&\qquad \{\mathcal{F}\{G_{a,\boldsymbol{\theta}}\}(-\varepsilon\boldsymbol{b})\}^\sim \, d^n\boldsymbol{b}\bigg) d\mu \\
&\overset{PT}{=} \int_{\mathbb{R}^+}\int_{S0(n)} \frac{1}{(2\pi)^n}\left(\int_{\mathbb{R}^n} F_{a,\boldsymbol{\theta}}(\boldsymbol{\xi})\widetilde{G_{a,\boldsymbol{\theta}}(\boldsymbol{\xi})}\, d^n\boldsymbol{\xi}\right) d\mu \\
&= \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n}\bigg(\int_{\mathbb{R}^+} a^n \int_{S0(n)} \hat{f}(\boldsymbol{\xi})\{\hat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\xi}))\}^\sim \\
&\qquad \hat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\xi}))\widetilde{\hat{g}(\boldsymbol{\xi})} d^n\boldsymbol{\xi}\bigg) d\mu \\
&= \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\xi}) \underbrace{\bigg(\int_{\mathbb{R}^+}\int_{S0(n)} a^n\{\hat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\xi}))\}^\sim \hat{\psi}(ar_{\boldsymbol{\theta}}^{-1}(\boldsymbol{\xi}))d\mu\bigg)}_{=C_\psi} \widetilde{\hat{g}(\boldsymbol{\xi})}\, d^n\boldsymbol{\xi} \\
&= \frac{1}{(2\pi)^n}\int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\xi})C_\psi\widetilde{\hat{g}(\boldsymbol{\xi})}\, d^n\boldsymbol{\xi} \\
&\overset{PT}{=} \int_{\mathbb{R}^n} f(\boldsymbol{x})C_\psi\widetilde{g(\boldsymbol{x})}\, d^n\boldsymbol{x} = (fC_\psi, g)_{L^2(\mathbb{R}^n;Cl_{n,0})},
\end{aligned} \tag{54}$$

where PT denotes the CFT Plancherel theorem. For the second equality we have also used the fact, that a substitution $\boldsymbol{b}' = -\varepsilon\boldsymbol{b}, \varepsilon = \pm 1$, as in $\int_{\mathbb{R}^n} h(-\varepsilon\boldsymbol{b})\, d^n\boldsymbol{b} = \int_{\mathbb{R}^n} h(\boldsymbol{b}')d^n\boldsymbol{b}'$, does not change the overall sign. □

As a corollary we get the following *norm relation*:

$$\|T_\psi f\|^2_{L^2(\mathcal{G};Cl_n)} = Sc(fC_\psi, f)_{L^2(\mathbb{R}^n;Cl_n)} \tag{55}$$

$$= C_\psi * (f,f)_{L^2(\mathbb{R}^n;Cl_n)}. \tag{56}$$

We can further derive the

Theorem 1 (Inverse Clifford Cl_n wavelet transform)**.** *Any $f \in L^2(\mathbb{R}^n; Cl_n)$ can be decomposed with respect to an admissible Clifford GA wavelet as*

$$\begin{aligned}
f(\boldsymbol{x}) &= \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\,\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}\,C_\psi^{-1}\, d\mu d^n\boldsymbol{b} \\
&= \int_{\mathcal{G}} (f,\psi_{a,\boldsymbol{\theta},\boldsymbol{b}})_{L^2(\mathbb{R}^n;Cl_n)}\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}C_\psi^{-1}\, d\mu d^n\boldsymbol{b},
\end{aligned} \tag{57}$$

the integral converging in the weak sense.

Proof. For any $g \in L^2(\mathbb{R}^n; Cl_{n,0})$

$$\begin{aligned}
&(T_\psi f, T_\psi g)_{L^2(\mathcal{G};Cl_{n,0})} \\
&= \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\{T_\psi g(a,\boldsymbol{\theta},\boldsymbol{b})\}^\sim\, d\mu d^n\boldsymbol{b} \\
&= \int_{\mathcal{G}}\int_{\mathbb{R}^n} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})\widetilde{g(\boldsymbol{x})}\, d^n\boldsymbol{x} d\mu d^n\boldsymbol{b} \\
&= \int_{\mathbb{R}^n}\int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})\, d\mu d^n\boldsymbol{b}\,\widetilde{g(\boldsymbol{x})}\, d^n\boldsymbol{x} \\
&= \left(\int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}\, d\mu d^n\boldsymbol{b},\, g\right)_{L^2(\mathbb{R}^n;Cl_{n,0})} \\
&\overset{IPR}{=} (fC_\psi\,,\, g)_{L^2(\mathbb{R}^n;Cl_{n,0})},
\end{aligned} \tag{58}$$

where IPR denotes the inner product relation (49). Because (58) holds for any $g \in L^2(\mathbb{R}^n; Cl_{n,0})$ we get

$$f(\boldsymbol{x})C_\psi = \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{b},\boldsymbol{\theta})\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})\, d\mu d^n\boldsymbol{b}, \tag{59}$$

or equivalently the *inverse CWT*

$$f(\boldsymbol{x}) = \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{b},\boldsymbol{\theta})\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})\, C_\psi^{-1}\, d\mu d^n\boldsymbol{b}. \tag{60}$$

□

Next is the *reproducing kernel*: We define for an admissible Clifford mother wavelet $\psi \in L^2(\mathbb{R}^n; Cl_n)$

$$\begin{aligned}
&\mathbb{K}_\psi(a,\boldsymbol{\theta},\boldsymbol{b}; a',\boldsymbol{\theta}',\boldsymbol{b}') \\
&\quad = \left(\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}C_\psi^{-1}, \psi_{a',\boldsymbol{\theta}',\boldsymbol{b}'}\right)_{L^2(\mathbb{R}^n;Cl_n)}.
\end{aligned} \tag{61}$$

Then $\mathbb{K}_\psi(a,\boldsymbol{\theta},\boldsymbol{b}; a',\boldsymbol{\theta}',\boldsymbol{b}')$ is a reproducing kernel in $L^2(\mathcal{G}, d\lambda)$, i.e,

$$\begin{aligned}
&T_\psi f(a',\boldsymbol{\theta}',\boldsymbol{b}') \\
&\quad = \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\mathbb{K}_\psi(a,\boldsymbol{\theta},\boldsymbol{b}; a',\boldsymbol{\theta}',\boldsymbol{b}')d\lambda.
\end{aligned} \tag{62}$$

Proof. By inserting for $f(\boldsymbol{x})$ the inverse CWT (57) into the definition of the CWT we obtain

$$\begin{aligned}
&T_\psi f(a',\boldsymbol{\theta}',\boldsymbol{b}') \\
&= \int_{\mathbb{R}^n}\left(\int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\,\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})\, d\lambda\, C_\psi^{-1}\right) \\
&\qquad \{\psi_{a',\boldsymbol{\theta}',\boldsymbol{b}'}(\boldsymbol{x})\}^\sim\, d^n\boldsymbol{x} \\
&= \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b}) \\
&\qquad \underbrace{\left(\int_{\mathbb{R}^n}\psi_{a,\boldsymbol{\theta},\boldsymbol{b}}(\boldsymbol{x})C_\psi^{-1}\{\psi_{a',\boldsymbol{\theta}',\boldsymbol{b}'}(\boldsymbol{x})\}^\sim\, d^n\boldsymbol{x}\right)}_{=\mathbb{K}_\psi} d\lambda \\
&= \int_{\mathcal{G}} T_\psi f(a,\boldsymbol{b},\boldsymbol{\theta})\mathbb{K}_\psi(a,\boldsymbol{\theta},\boldsymbol{b}; a',\boldsymbol{\theta}',\boldsymbol{b}')\, d\lambda.
\end{aligned} \tag{63}$$

□

Theorem 2 (Generalized Clifford GA wavelet uncertainty principle)**.** *Let ψ be an admissible Clifford algebra mother wavelet. Then for every $f \in L^2(\mathbb{R}^n; Cl_n)$, the following inequality holds*

$$\|\boldsymbol{b}T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\|^2_{L^2(\mathcal{G};Cl_n)} C_\psi * (\widetilde{\boldsymbol{\omega}\hat{f}}, \widetilde{\boldsymbol{\omega}\hat{f}})_{L^2(\mathbb{R}^n;Cl_n)} \geq \frac{n(2\pi)^n}{4}\left[C_\psi * (f,f)_{L^2(\mathbb{R}^n;Cl_n)}\right]^2. \quad (64)$$

NB: The integrated variance

$$\int_{\mathbb{R}^+}\int_{SO(n)} \|\boldsymbol{\omega}\mathcal{F}\{T_\psi f(a,\boldsymbol{\theta},\,.\,)\}\|^2_{L^2(\mathbb{R}^n;Cl_n)} d\mu \quad (65)$$

is *independent* of the wavelet parity ε. Otherwise the proof is similar to the one for $n=3$ in [16]. For scalar admissibility constant this reduces to

Corollary 1 (Uncertainty principle for Clifford GA wavelet)**.** *Let ψ be a Clifford algebra wavelet with scalar admissibility constant $C_\psi \in \mathbb{R}^n$. Then for every $f \in L^2(\mathbb{R}^n; Cl_n)$, the following inequality holds*

$$\|\boldsymbol{b}T_\psi f(a,\boldsymbol{\theta},\boldsymbol{b})\|^2_{L^2(\mathcal{G};Cl_n)}\ \|\boldsymbol{\omega}\hat{f}\|^2_{L^2(\mathbb{R}^n;Cl_n)} \geq nC_\psi \frac{(2\pi)^n}{4}\|f\|^4_{L^2(\mathbb{R}^n;Cl_n)}. \quad (66)$$

- This shows indeed, that Theorem 2 represents a multivector generalization of the uncertainty principle for Clifford wavelets with scalar admissibility constant.
- Compare with the (direction independent) uncertainty principle (28) for the CFT.

3.3 Example of Clifford GA Gabor wavelets

Finally *Clifford (geometric) algebra Gabor Wavelets* are defined as (variances $\sigma_k, 1 \leq k \leq n$, for $n = 2(\text{mod } 4) : A \in Cl_n^+$ or $A \in Cl_n^-$)

$$\psi^c(\boldsymbol{x}) = \frac{A\, e^{-\frac{1}{2}\sum_k \frac{x_k^2}{\sigma_k^2}}}{(2\pi)^{\frac{n}{2}}\prod_{k=1}^n \sigma_k}\,(e^{i_n\boldsymbol{\omega}_0\cdot\boldsymbol{x}} - \underbrace{e^{-\frac{1}{2}\sum_{k=1}^n \sigma_k^2\omega_{0,k}^2}}_{\text{constant}}),$$
$$\boldsymbol{x},\boldsymbol{\omega}_0 \in \mathbb{R}^n, \quad \text{constant } A \in Cl_n\,. \quad (67)$$

The spectral (CFT) representation of the Clifford Gabor wavelets (67) is

$$\mathcal{F}\{\psi^c\}(\boldsymbol{\omega}) = \widehat{\psi^c}(\boldsymbol{\omega}) \quad (68)$$
$$= A(\underbrace{e^{-\frac{1}{2}\sum_k \sigma_k^2(\omega_k-\omega_{0,k})^2} - e^{-\frac{1}{2}\sum_k \sigma_k^2(\omega_k^2+\omega_{0,k}^2)}}_{=\phi(\boldsymbol{\omega})\in\mathbb{R}}).$$

It follows with (33) that the Clifford Gabor wavelet admissibility constant

$$C_{\psi^c} = \int_{\mathbb{R}^n} \frac{\{\widehat{\psi^c}(\boldsymbol{\xi})\}^\sim \widehat{\psi^c}(\boldsymbol{\xi})}{|\boldsymbol{\xi}|^n}\, d^n\boldsymbol{\xi} = \widetilde{A}A\int_{\mathbb{R}^n}\frac{\phi(\boldsymbol{\xi})^2}{|\boldsymbol{\xi}|^n}\, d^n\boldsymbol{\xi}\,. \quad (69)$$

If e.g. A is a vector or a product of vectors (versor), then C_{ψ^c} will be scalar.

4 Conclusion

We have introduced real Clifford (geometric) algebra wavelets for multivector signals taking values in Cl_n. *Real* means that we completely avoid to use the field of complex numbers $\mathbb{C}$. This also applies to the use of a real Clifford (geometric) algebra Fourier transform for the spectral representation. An extension to $Cl_{0,n'}, n' = 1,2(\text{mod } 4)$ appears straight forward.

Acknowledgments

Soli deo gloria. I do thank my dear family, B. Mawardi, G. Scheuermann, D. Hildenbrand and V. Skala.

References

[1] P. Goupillaud, *Biography of Jean P. Morlet*, `http://www.mssu.edu/seg-vm/bio_jean_p__morlet.html`

[2] M. Mitrea, Clifford Wavelets, Singular Integrals and Hardy Spaces. *Lect. Notes in Math.* **1575**, Springer, New York, 1994.

[3] Brackx et al., *Ghent Clifford Analysis Wavelet Publications* (2001-2007), `http://cage.ugent.be/crg/cliffordpublicaties.PDF`

[4] L. Traversoni, Quaternion Wavelet Problems. *Proc. of 8th Int. Symp. on Approx. Theory*, Texas A&M University, Jan. 1995. Image Analysis Using Quaternion Wavelets. in E. Bayro-Corrochano, G. Sobczyk (eds.), *Proc. of AGACSE 1999*, Birkhäuser, Basel, 2001.

[5] E. Bayro-Corrochano, *List of Publications*, `http://www.gdl.cinvestav.mx/edb/publications.html`

[6] J. Zhao, and L. Peng. Quaternion-valued Admissible Wavelets Associated with the 2D Euclidean Group with Dilations. *J. of Nat. Geom.*, 2001; J. Zhao. Clifford algebra-valued Admissible Wavelets Associated with Admissible Group. *Acta Sci. Nat. Univ. Pek.*, **41**(5), (2005).

[7] Kähler et al. Monogenic Wavelets over Unit Ball. *ZAA*, **24**, 813–824 (2005) .

[8] S. Bernstein, T.E. Simos et al. (eds.). Clifford Continuous Wavelet Transforms in $L_{0,2}$ and $L_{0,3}$. *AIP Proc. of ICNAAM 2008*, **1048**, 634–637 (2008); S. Bernstein and S. Ebert, Spherical Wavelets, Kernels and Symmetries. *AIP Proc. of ICNAAM 2009*, **1168**, 761–764 (2009); S. Bernstein, S. Ebert and R.S. Krausshar, Diffusion Wavelets on Conformally Flat Cylinders and Tori. *AIP Proc. of ICNAAM 2009*, **1168**, 773–776 (2009); S. Bernstein, Spherical Singular Integrals, Monogenic Kernels and Wavelets on the 3D Sphere. *AACA*, **19**(2), 173–189 (2009).

[9] E. Hitzer and R. Abłamowicz, Geometric Roots of -1 in Clifford Algebras $Cl(p,q)$ with $p+q \leq 4$. Submitted to *Adv. App. Cliff. Alg.*, May 2009. Preprint version: Technical Report 2009-3, Department of Mathematics, Tennessee Technological University, Tennessee, USA, May 2009. Also available as: arXiv:0905.3019v1 [math.RA]

[10] B. Mawardi and E. Hitzer, Clifford Fourier Transformation and Uncertainty Principle for the Clifford Geometric Algebra $Cl(3,0)$. *Adv. App. Cliff. Alg.*, **16**(1), 41–61 (2006).

[11] E. Hitzer and B. Mawardi, Clifford Fourier Transform on Multivector Fields and Uncertainty Principles for Dimensions $n = 2 \,(\mathrm{mod}\, 4)$ and $n = 3 \,(\mathrm{mod}\, 4)$. *Adv. App. Cliff. Alg.* Vol. **18**(S3,4), 715–736 (2008).

[12] B. Mawardi, E. Hitzer and S. Adji, Two-Dimensional Clifford Windowed Fourier Transform. acc. for G. Scheuermann, E. Bayro-Corrochano (eds.), *Appl. Geom. Algs. in Comp. Sc. and Engineering*, Springer, New York, 2010.

[13] T. A. Ell, Quaterion-Fourier Transform for Analysis of Two-dimensional Linear Time-Invariant Partial Differential Systems. *Proc. 32nd IEEE Conf. on Decision and Control*, Dec. 1993, 1830–1841.

[14] E. Hitzer, Quaternion Fourier Transform on Quaternion Fields and Generalizations. *Adv. App. Cliff. Alg.*, **17**(3), 497–517 (2007); E. Hitzer, Directional Uncertainty Principle for Quaternion Fourier Transforms. *Adv. App. Cliff. Alg.*, Online First, 14 pp. (2009).

[15] B. Mawardi, E. Hitzer, R. Ashino and R. Vaillancourt, Windowed Fourier transform of two-dimensional quaternionic signals. Submitted to *Appl. Math. and Comp.*, March 2009.

[16] B. Mawardi and E. Hitzer, Clifford Algebra $Cl_{3,0}$-valued Wavelet Transformation, Clifford Wavelet Uncertainty Inequality and Clifford Gabor Wavelets. *Int. J. of Wavelets, Multires. and Inf. Proces.*, **5**(6), 997–1019 (2007).

[17] J. Ebling and G. Scheuermann, Clifford Fourier Transform on Vector Fields. *IEEE Trans. on Vis. and Comp. Graph.*, **11**(4), (July/Aug. 2005); Clifford Convolution And Pattern Matching On Vector Fields. *Proc. 14th IEEE Vis. 2003 (VIS'03)*, p. 26, (Oct. 22-24, 2003).

[18] E. Hitzer, T.E. Simos et al. (eds.). Real Clifford Algebra $Cl_{n,0}, n = 2, 3(\mathrm{mod}\ 4)$ Wavelet Transform. *AIP Proc. of ICNAAM 2009*, **1168**, 781–784 (2009).

[19] E. Hitzer and B. Mawardi, Uncertainty Principle for the Clifford Geometric Algebra $Cl(3,0)$ based on Clifford Fourier Transform. in TE. Simos, G. Sihoyios, C. Tsitouras (eds.), *Proc. of ICNAAM 2005*, Wiley-VCH, Weinheim, 922–925 (2005).

[20] E. Hitzer and B. Mawardi, Uncertainty Principle for Clifford Geometric Algebras $Cl_{n,0}, n = 3(\mathrm{mod}\ 4)$ based on Clifford Fourier Transform. in T. Qian, M.I. Vai, X. Yusheng (eds.), *Wavelet Analysis and Applications, Springer (SCI) Book Series Applied and Numerical Harmonic Analysis*, Springer, 45–54 (2006).

[21] B. Mawardi, E. Hitzer, A. Hayashi and R. Ashino, An Uncertainty Principle for Quaternion Fourier Transform, *Comp. & Math. with Appl.*, **56**, 2398–2410 (2008).

A Study of 3-D Surface Registration Using Distance Map and 3-D Radon Transform

Makoto Hasegawa
Faculty of Engineering, Kinki University
1 Takayaumenobe, Higashihiroshima, Hiroshima, 739-2116 Japan
hasegawa@hiro.kindai.ac.jp

ABSTRACT

A three-dimensional surface model registration using distance map and three-dimensional Radon transform is proposed. Phase-only matched filter can be applied to the three-dimensional surface model registration just like conventional two-dimensional image registration. Our registration procedure is described, and we execute simulations.

Keywords
Surface model registration, distance map, Radon transform, R-transform, phase-only matched filter

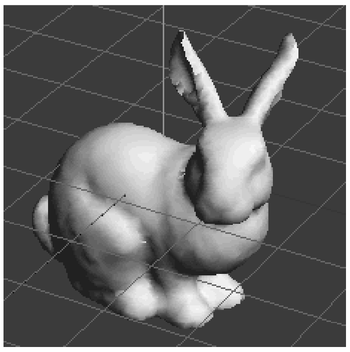
1. INTRODUCTION

Three-dimensional surface model registration is an important and fundamental topic in computer graphics. The surface model registration means location matching between a template surface model and a target surface model described many polygons and vertices. Various registration methods are proposed. Chen and Medioni propose the iterative closest point algorithm [Che91]. In their algorithm, the combination of common vertex between target and template surface model is assumed the closest pair. The target surface model is displaced by the minimizing the location gap between the corresponded vertex pair. Such a process is iterated until the location gap becomes small. It is supposed that much iteration lead high computational complexity. Because the solution is oscillated, the stable registration can not be obtained in the iteration.

By the way, two-dimensional image registrations are obviously different. Phase-only matched filter proposed by Kuglin and Hines is a conventional method for the two-dimensional image recognition [Kug75]. We can use the phase-only matched filter for the two-dimensional image registration. The phase-only matched filter uses Fourier transform, and the object location gap in the images is detected. We translate the objects by the detected location gap. By the way, when the objects are not only translated but also rotated, the ordinary phase-only matched filter is not available. Then, Chen and his co-workers propose a method using the phase-only matched filter and Fourier-Mellin transform, which is effective for the rotation registration [Che94], where the Fourier-Mellin transform is proposed by Sheng and Duvernoy [She86]. In addition, we proposed a two-dimensional image registration using the phase-only matched filter and Radon transform [Has07, Has09]. The phase-only matched filter doesn't execute iteration process, and we can get the solution directly. In addition, the sub-pixel level registration is possible. However, we can not apply the phase-only matched filter to the three-dimensional surface model registration. Because, polygon vertices don't align regularly like a lattice structure, the Fourier transform is impossible. Therefore, the phase-only matched filter is not available for the three-dimensional surface model registration.

We propose a novel three-dimensional surface model registration using the phase-only matched filter without the iteration process. We define a distance map in our method. First, we consider a three-dimensional space include the surface model. On each pixel in the space, we calculate the distance with the closest vertex of the surface model. Each pixel is attached the inverse value of the distance. The distance map is such the three-dimensional space, which is a three-dimensional volume data. Because the distance map has a lattice structure, the Fourier transform is available. Actually, we create two-dimensional projection images (x-y projection image and z-y projection image). Then, we can execute the two-dimensional Fourier transform to the projection images. Therefore, a general two-dimensional phase-only matched filter can be applied into our three-dimensional surface model registration just like conventional two-dimensional image registration.

When the surface model is rotated, we have to execute registration for the rotation. We apply R-transform proposed by Tabbone and his do-workers [Tab06]. In their method, they deal with the two-dimensional image recognition. First, they convert a two-dimensional image using Radon transform. The


(a)

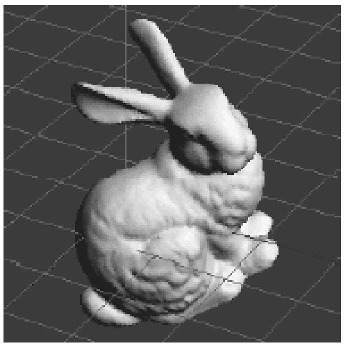
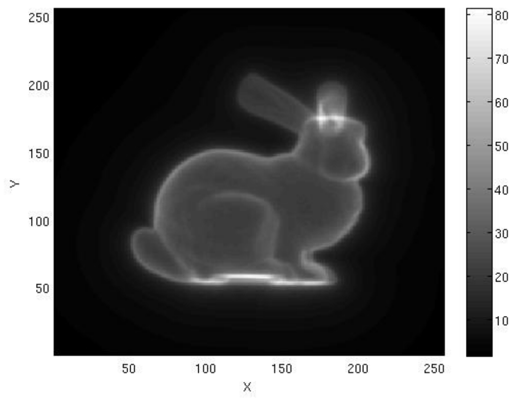

(b)

Figure 1: Surface model. (a) "Stanford Bunny," (b) "Rotated and transformed surface model (Bunny2)."

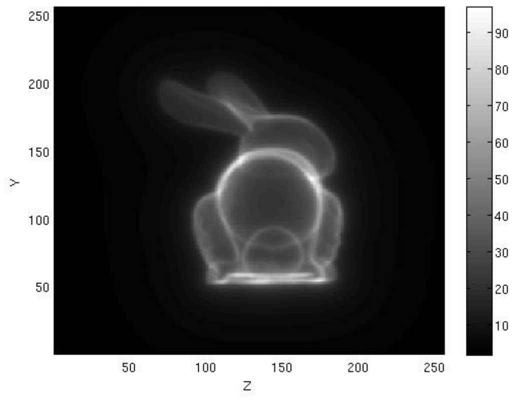
result of Radon transform is expressed using the polar coordinate system. They calculate square integration along radius on each argument, and a one-dimensional R-transform signal is obtained. When the object is rotated in the image, the R-transform signal is shifted according to the rotation angle. Therefore, the rotation registration can be executed by the R-transform signal comparison.

By the way, our registration is for the three-dimensional surface models. We convert the distance maps of template and target surface model using three-dimensional Radon transform, and we obtain a three-dimensional Radon volume data. After that, we execute the R-transform to the three-dimensional Radon volume data, and we obtain the two-dimensional R-transform images. When the surface models are rotated, the R-transform images are changed according to the rotation angle. We rotated the surface models so that the R-transform images become equal. After the rotation registration, we execute the translation registration using the phase-only matched filter.

In the following section, we define our distance map. The three-dimensional Radon transform and R-transform are explained in Section 3. We describe our



(a)



(b)

Figure 2: Distance map of "Stanford Bunny." (a) x-y projection image and (b) z-y projection image.

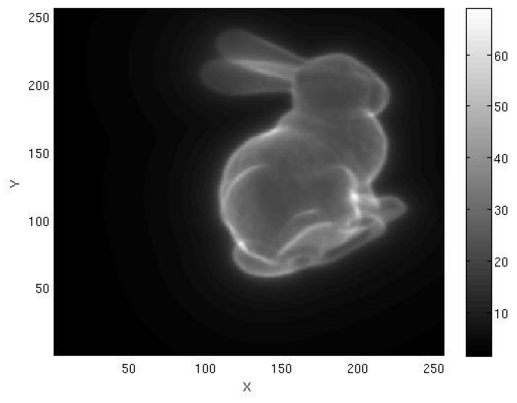
registration procedure in Section 4. The result of a simulation of our registration is shown in Section 5.

2. DISTANCE MAP

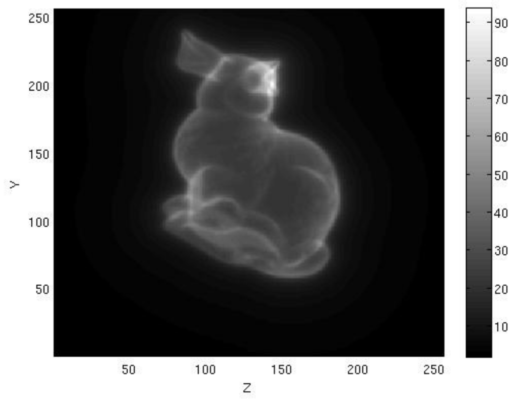
A coordinate in the three-dimensional space is represented as $\mathbf{x} = (x, y, z)$. We define a distance map $f(\mathbf{x})$ as

$$f(x) = \frac{1}{\min_{v_i \in S}(\|\mathbf{x} - \mathbf{v}_i\|)}, \qquad (1)$$

where S is a surface model, v_i is a polygon vertex of the surface model S, $\mathbf{v}_i$ is a coordinate of the polygon vertex v_i, and $\|\mathbf{x}\| = \sqrt{x^2 + y^2 + z^2}$. Hence, each pixel value of the distance map $f(\mathbf{x})$ is the inverse value of the distance to the closest vertex of the surface model S.



(a)



(b)

Figure 3: Distance map of "Bunny2": (a) x-y projection image and (b) z-y projection image.

The surface model "Stanford Bunny" is shown in Figure 1 (a). It has 34834 vertices in the surface model. The surface model "Bunny2" shown in Figure 1 (b) is a rotated and translated of "Stanford Bunny."

The distance maps of "Stanford Bunny" and "Bunny2" are shown in Figure 2 and Figure 3, respectively. The volume data of the distance maps is composed by $256 \times 256 \times 256$ pixels. Because the distance map has a lattice structure, the Fourier transform is available. Therefore, the phase-only matched filter can be applied into our three-dimensional surface model registration.

In addition, to improve the processing speed, we deal with the two-dimensional projection images. The x-y plain projection images are shown in Figure 2 (a) and Figure 3 (a), and the z-y plain projection images are shown in Figure 2 (b) and Figure 3 (b). We can execute the two-dimensional general phase-only matched filter to the projection images for three-dimensional surface model registration.

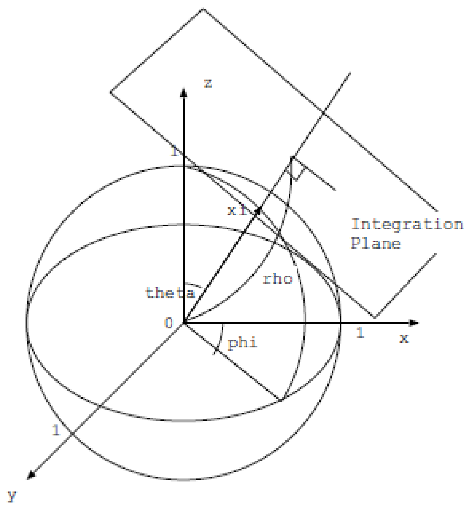


Figure 4: Three-dimensional Radon transform.

3. 3-D RADON TRANSFORM AND R-TRANSFORM

When the surface model is rotated, we have to execute the registration for the rotation. We apply three-dimensional Radon transform and R-transform.

We convert the distance map $f(\mathbf{x})$ using three-dimensional Radon transform. The three-dimensional Radon transform is defined as

$$T_f(\phi, \theta, \rho) = \int f(\mathbf{x}) \delta(\mathbf{x}^T \xi - \rho) d\mathbf{x}, \quad (2)$$

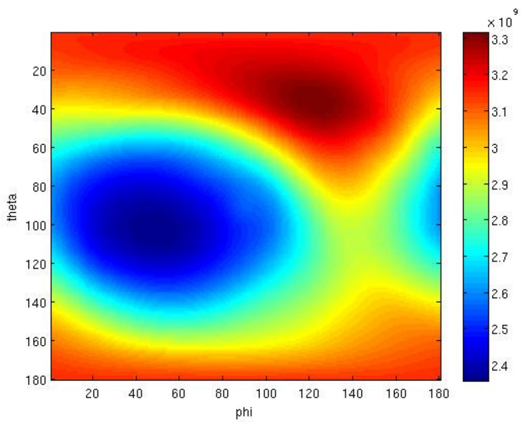
where $\xi = (\cos\phi \sin\theta, \sin\phi \cos\theta, \cos\theta)$. The range of the argument ϕ is $0 \le \phi < \pi$, and the range of the argument θ is $0 \le \theta < \pi$. The three-dimensional Radon transform means plane integration in the three-dimensional volume data as Figure 4. The integration path is a two-dimensional plane which is orthogonal with the vector ξ , and the distance from the origin is ρ .

We convert the three-dimensional Radon volume data using R-transform. The R-transform is defined as

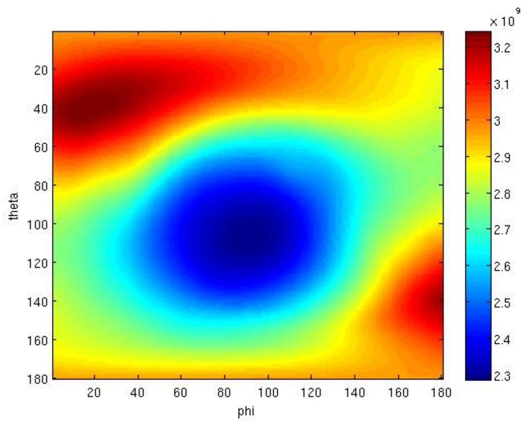
$$R_f(\phi, \theta) = \int T^2{}_f(\phi, \theta, \rho) d\rho . \quad (3)$$

The result of the R-transform is expressed using the arguments ϕ and θ in an orthogonal coordinates system. Therefore, we obtain two-dimensional R-transform images.

The R-transform image of "Stanford Bunny" and "Bunny2" are shown in Figure 5 (a) and (b), respectively. When the surface model is translated, the R-transform image does not change. However,



(a)



(b)

Figure 5: R-transform image: (a) "Stanford Bunny" and "Bunny2."

when the surface model is rotated, the R-transform image changes according to the rotation angle. In the case of rotation centering on Z axis (only ϕ degree rotation), the R-transform image is translated horizontally. When the surface model is rotated like that the North Pole shifts from z axis (θ degree rotation), the R-transform image changes relatively complexly. Then, we rotated the surface models so that the R-transform images become equal.

4. REGISTRATION PROCEDURE

Our registration procedure is shown as follows. The target surface model is described as f , and the template model is described as g .

[Step 1] Create distance map of the two surface model f and g . The f 's distance map is $f(\mathbf{x})$, and the g 's distance map is $g(\mathbf{x})$.

[Step 2] Convert the distance map using three-dimensional Radon transform. We obtain a Radon volume data $T_f(\rho,\phi,\theta)$ and $T_g(\rho,\phi,\theta)$.

[Step 3] Convert the Radon volume data using R-transform. We obtain an R-transform image $R_f(\phi,\theta)$ and $R_g(\phi,\theta)$. These images are shown in Figure 5 (a) and Figure 5 (b), respectively.

[Step 4] Rotate the surface models so that the minimum position in the R-transform image become $(\phi,\theta)=(0,0)$. The minimum position is moved to the North Pole in Figure 4 (at the same time, the minimum position is moved to the South Pole).

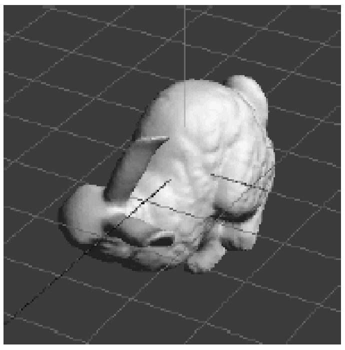
1) We look for the minimum position (ϕ_f,θ_f) in the R-transform image $R_f(\phi,\theta)$.

2) We rotate the surface model f anti-clockwise by ϕ_f degrees centering on Z axis, so that the minimum position in the R-transform image $R_f(\phi,\theta)$ becomes $(0,\theta_f)$.

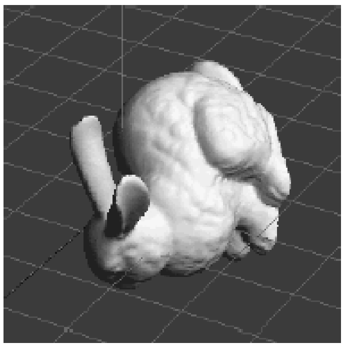
3) We rotate the surface model f anti-clockwise by θ_f degrees centering on Y axis, so that the minimum position in the R-transform image $R_f(\phi,\theta)$ becomes $(0,0)$.

We also rotate the surface model g in a similar way. Consequently, the R-transform images $R_f(\phi,\theta)$ and $R_g(\phi,\theta)$ become equal excluding the rotation gap centering on Z axis (ϕ rotation).

[Step 5] Calculate the correlation between $R_f(\phi,\theta)$ and $R_g(\phi,\theta)$. The correlation $C(\phi,\theta)$ is defined as,



(a)



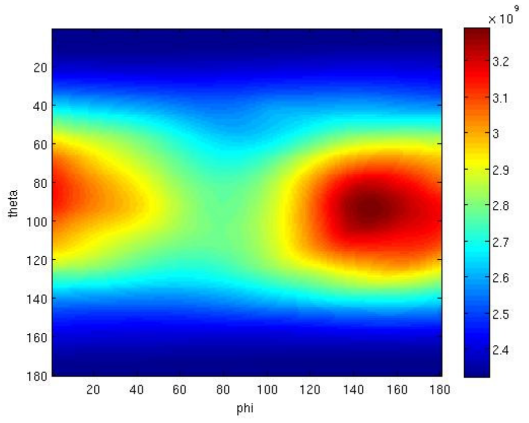
(b)

Figure 6: Rotation correlation. (a) "Stanford Bunny," (b) "Bunny2."

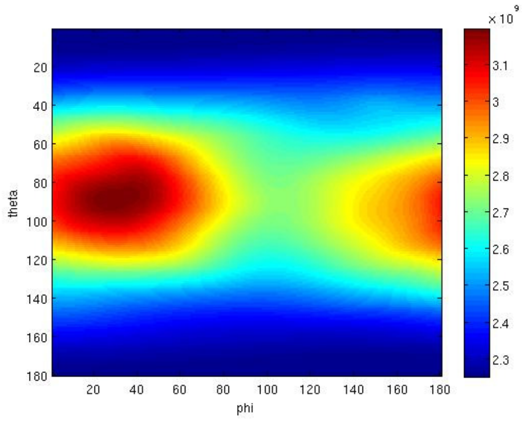
$$\hat{R}_f(u,v) = \iint R_f(\phi,\theta) e^{-i(u\phi+v\theta)} d\phi d\theta,$$
$$C(\phi,\theta) = \frac{1}{(2\pi)^2} \iint \hat{R}_f(u,v)\hat{R}_g(u,v) e^{i(u\phi+v\theta)} dudv. \tag{4}$$

Find the maximum position (ϕ_c, θ_c) in the correlation $C(\phi,\theta)$. Rotate the surface model f anti-clockwise by ϕ_c degrees centering on Z axis. Therefore, the R-transform images $R_f(\phi,\theta)$ and $R_g(\phi,\theta)$ become equal. The two surface models' mis-registration becomes only a translation gap.

[Step 6] Re-create the distance maps $f(\mathbf{x})$ and $g(\mathbf{x})$. Create x-y and y-z projection images $f_{xy}(x,y)$, $f_{zy}(z,y)$, $g_{xy}(x,y)$, and $g_{zy}(z,y)$.



(a)



(b)

Figure 7: Rotation correlation of the R-transform image: (a) "Stanford Bunny" and "Bunny2."

[Step 7] Execute the Phase-only matched filter to the projection images $f_{xy}(x,y)$ and $g_{xy}(x,y)$ in order to detect the translation gap between two surface models f and g. The result of the phase-only matched filter $POC(x,y)$ is defined as,

$$\hat{f}_{xy}(u,v) = \iint f_{xy}(x,y) e^{-i(ux+vy)} dxdy,$$
$$POC(x,y) = \frac{1}{(2\pi)^2} \iint \hat{f}_{xy}(u,v)\hat{g}_{xy}(u,v) e^{i(ux+vy)} dudv. \tag{5}$$

The phase-only correlation $POC(x,y)$ has a sharp peak. The peak position shows the translation gap for X axis and Y axis. Similarly, we execute the Phase-only matched filter to the projection images $f_{zy}(z,y)$ and $g_{zy}(z,y)$. We also detect translation

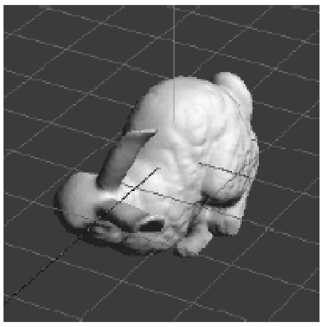


Figure 8: Re-rotated "Bunny2."

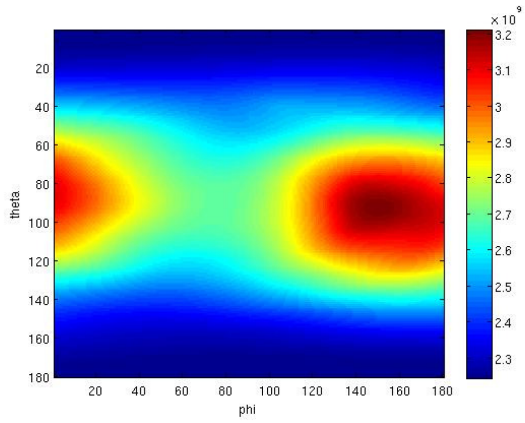


Figure 9: R-transform image of the re-rotated "Bunny2."

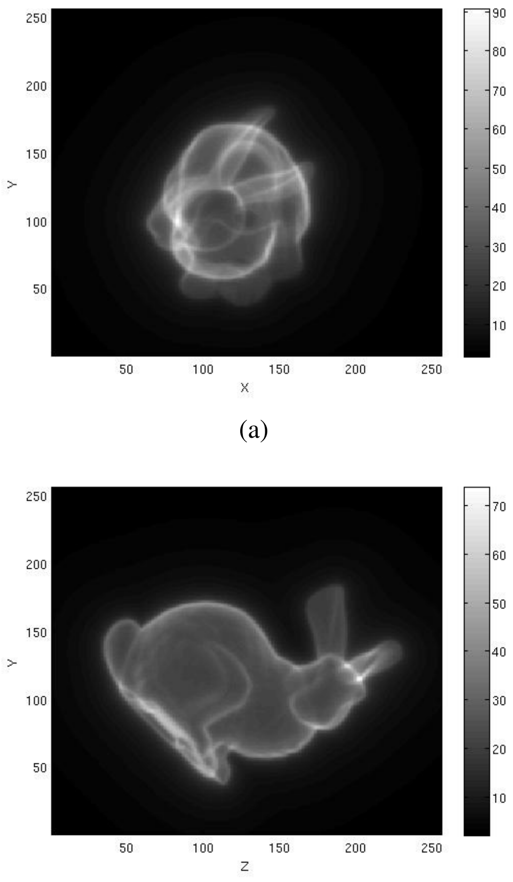


Figure 10: Distance map of the rotation correlated"Stanford Bunny": (a) x-y projection image and (b) z-y projection image.

gap for Z axis. We translate the target surface model f , and our registration is competed.

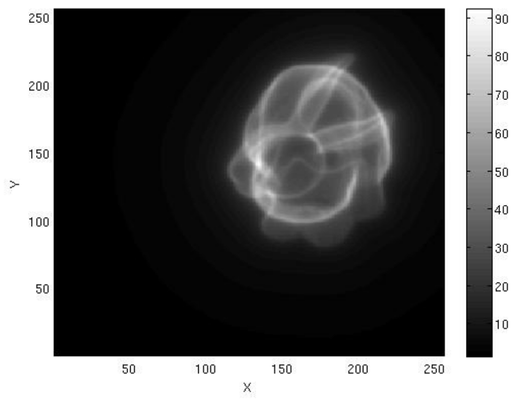
5. EXPERIMETAL RESULTS

We execute a simulation for our surface model registration. The template model is "Stanford Bunny" and the target model is "Bunny2." These surface models are shown in Figure 1 (a) and (b), respectively. We rotate "Bunny2" anti-clockwise by 30 degrees centering on Z axis; rotate anti-clockwise by 45 degrees centering on Y axis; translate 10, 20, and -10 pixels for X, Y, and Z axis, respectively. Then, we can get the target model "Bunny2."

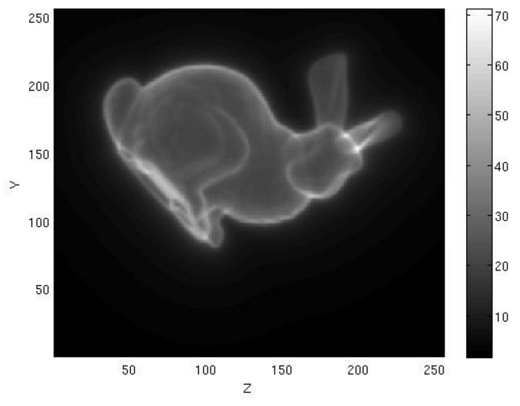
First, we create distance maps of "Stanford Bunny" and "Bunny2." The x-y projection images of "Stanford Bunny" and "Bunny2" are shown in Figure 2 (a) and Figure 3 (a); and the z-y projection images are shown in Figure 2 (b) and Figure 3 (b), respectively.

Next, we convert the distance maps using three-dimensional Radon transform and R-transform. We obtain two R-transform images of "Stanford Bunny" and "Bunny2," shown in Figure 5 (a) and (b), respectively. The minimum position (ϕ, θ) of the R-transform image of "Stanford Bunny" is (47, 101). Then, we rotate the surface model "Stanford Bunny" anti-clockwise by 47 degrees centering on Z axis; rotate anti-clockwise by 101 degrees centering on Y axis. Therefore, the surface model "Stanford Bunny" become as Figure 6 (a), and the R-transform image become as Figure 7 (a). Similarly, the minimum position (ϕ, θ) of the R-transform image for "Bunny2" is (91, 105). Then, we rotate the surface model "Bunny2" anti-clockwise by 91 degrees centering on Z axis; rotate anti-clockwise by 105 degrees centering on Y axis. Therefore, the surface model "Bunny2" become as Figure 6 (b), and the R-transform image become as Figure 7 (b).

Consequently, the two R-transform images of "Stanford Bunny" and "Bunny2" are equal excluding the rotation gap for the argument ϕ . We execute the



(a)

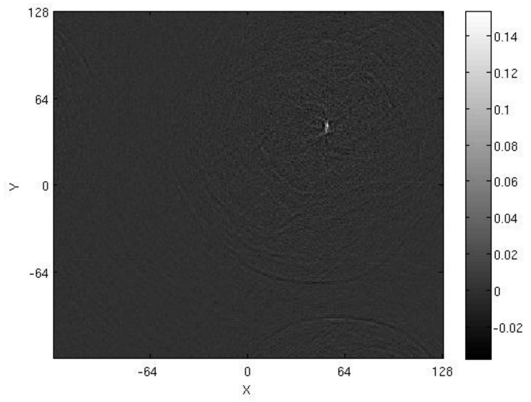


(b)

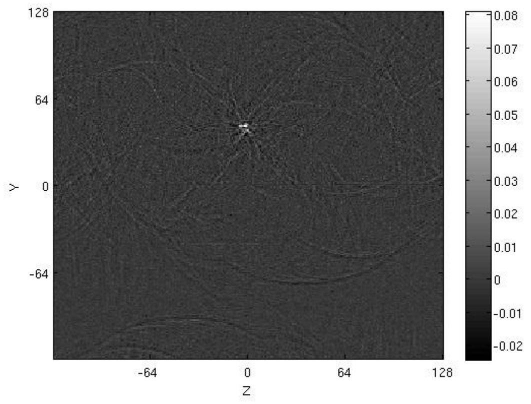
Figure 11: Distance map of the re-rotation correlated"Stanford Bunny": (a) x-y projection image and (b) z-y projection image.

phase-only matched filter to these R-transform images, and we detect the rotation gap which is 46 degree for the argument ϕ. We rotate the surface model "Bunny2" anti-clockwise by 46 degrees centering on Z axis. Therefore, the surface model "Bunny2" become as Figure 8. The surface model "Stanford Bunny" and "Bunny2" become equal excluding the translation gap. In addition, the R-transform images for "Stanford Bunny" and "Bunny2" become equal as Figure 7 (a) and Figure 9.

We re-create the distance maps for "Stanford Bunny" and "Bunny2." The x-y projection images of "Stanford Bunny" and "Bunny2" are shown in Figure 10 (a) and Figure 11 (a); and the z-y projection images are shown in Figure 10 (b) and Figure 11 (b), respectively. We execute the phase-only matched filter to the x-y projection images as Figure 10 (a) and Figure 11 (a), and we obtain a phase-only correlation image as Figure 12 (a). In the correlation image of Figure 12 (a), we have a sharp peak which is located at (52, 43). The peak location means the



(a)



(b)

Figure 12: Translation correlation using phase-only matched filter: (a) x-y projection and (b) z-y projection.

translation gap as 52 pixels for X axis and 43 pixels for Y axis. We translate the surface model "Bunny2" to 52 pixels for X axis, 43 pixels for Y axis. Similarly, we execute the phase-only matched filter to the z-y projection images Figure 10 (b) and Figure 11 (b), and we obtain a phase-only correlation image as Figure 12 (b). In the correlation image of Figure 12 (b), we have a sharp peak which is located at (-3, 43). The peak location means the translation gap as -3 pixels for Z axis. We translate the surface model "Bunny2" to -3 pixels for Z axis.

The rotation and translation registration between "Stanford Bunny" and "Bunny" is completed.

We show the matching scores which are values from the phase-only matched filter in Step 7 as Table 1. The surface models "Dragon" and "Happy Buddha" are used in the experiment. It is possible to use our approach for object identification.

Table 1: Matching Score

	Stanford Bunny	Dragon	Happy Buddha
Stanford Bunny	1.0000	0.0282	0.0323
Bunny2	0.2358	0.0273	0.0328
Dragon		1.0000	0.0311
Happy Buddha			1.0000

6. CONCLUSION

A method of surface model registration using distance map, three-dimensional Radon transform, and R-transform is proposed. We define the distance map which each pixel are attached an inverse value of the distance with the nearest polygon vertex. Because our distance space has a lattice structure, the Fourier transform is available. Therefore, the phase-only matched filter can be applied into our three-dimensional surface model registration just like conventional two-dimensional image registration. In addition, we create two-dimensional projection images (x-y projection, and z-y projection) from the three-dimensional distance map. We can apply two-dimensional phase-only matched filter to the projection images for the three-dimensional surface model registration.

For the rotation registration, we apply three-dimensional Radon transform and R-transform. We convert the distance space using three-dimensional Radon transform. After that, we convert using the R-transform, and we obtain a two-dimensional R-transform image. We rotated the target surface model so that the R-transform images of surface models become equal.

We executed a simulation using a surface model “Stanford Bunny”, and the possibility of our registration method was shown. In our future works, our registration accuracy and the computational complexity will be discussed. We will compare with other methods like the iterative closest point algorithm.

ACKNOWLEDGEMENTS

This research was supported by Takahashi Industrial and Economic Research Foundation.

REFERENCES

[Che91] Y. Chen and G. Medioni, “Object modeling by registration of multiple range images,” *IEEE Int. Conference on Robotics and Automation*, vol. 3, pp. 2724-2729, 1991.

[Kug75] C. Kuglin and D. Hines, “The pase correlation image alignment method,” *Proc. Int. Conf. on Cybernetics and Society*, pp.163-165, 1975.

[Che94] Q. Chen, M. Defrise, and F. Deconinck, “Symmetric phase-only matched filtering of Fourier-Mellin transform for image registration and recognition,” *IEEE Trans. PAMI*, vol. 16, no. 12, pp. 1156-1168, Dec. 1994.

[She86] Y. Shen and J. Duvernoy, “Circular-Fourier-radial-Mellin transform discriptors for pattern recogmition,” *J. Opt. Soc.*AM. A, vol. 3, no. 6, June 1986.

[Has07] M. Hasegawa, “Propasal of pattern matching using log-autocorrelation on Radon transform,” *Proc. of International Symposium on Communications and Information Technologies 2007 (ISCIT2007)*, Oct. 2007.

[Has09] M. Hasegawa, "Proposal of Amplitude only Logarithmic Radon Transform for Pattern Matching - Relation with Fourier-Mellin Transform -," *Proc. of International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS2008)*, Feb. 2009.

[Tab06] S. Tabbone, L. Wendling, and J. –P. Salmon, “A new shape descriptor defined the Radon transform,” *Computer Vision and Image understanding*, 102, pp.42-51, 2006.

Geometric Algebra Computing on the CUDA Platform

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

Andreas Görlitz
TU Darmstadt, Germany
Department of Computer Science
A.Goerlitz@stud.tu-darmstadt.de

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

ABSTRACT

Geometric Algebra (GA) is a mathematical framework that allows a compact, geometrically intuitive description of geometric relationships and algorithms. These algorithms require significant computational power because of the intrinsically high dimensionality of geometric algebras. Algorithms in an n-dimensional GA require 2^n elements to be computed for each multivector. GA is not restricted to a maximum of dimensions, so arbitrary geometric algebras can be constructed over a vector space $\mathcal{V}_n$. Since computations in GA can be highly parallelized, the benefits of a parallel computing architecture can lead to a significant speed-up compared to standard CPU implementations, where elements of the algebra have to be calculated sequentially. An upcoming approach of coping with parallel computing is to use general-purpose computation on graphics processing units (GPGPU). In this paper, we focus on the Compute Unified Device Architecture (CUDA) from NVIDIA [9]. We present a code generator that takes as input the description of an arbitrary geometric algebra and produces an implementation of geometric products for the underlying algebra on the CUDA platform.

Keywords: Geometric Algebra, Geometric Computing, GPU, CUDA.

1 INTRODUCTION

Geometric Algebra (GA) has become more and more popular in different fields of research. Using GA makes it possible to develop very compact algorithms while keeping them geometrically intuitive. One major drawback is the reduced performance when executing GA algorithms without further processing. But recent research has shown that it is possible to speed up GA algorithms drastically by means of static code optimization and switching to parallel computing architectures like field-programmable gate arrays (FPGA), for instance. Moreover, this can lead to performance improvements compared to standard implementations [8].

Applications written in GA require a very large number of calculations to be processed, e.g. feature extraction algorithms [11]. In many cases it is necessary to define highly customized non-standard algebras in order to fit the problem statement. What these problems have in common is a remarkable amount of parallelization required to fulfill the constraints of reduction. In theory, it is possible to decrease the order of time complexity for certain applications which, in turn, requires virtually infinite operations to be executed in parallel.

In this paper, we investigate the potential of executing GA operations on parallel architectures. As a very first approach we focus on the implementation of arbitrary geometric products on the CUDA platform as a means for evaluating the performance of parallel computing in GA compared to standard implementations. We implement the calculation of the geometric product without any restrictions to the underlying algebra and associated metric and signature. We exploit the property that elements of the result multivector can be easily computed in parallel, e.g. each one in a separate (parallel) thread on a CUDA-enabled GPU. Therefore, we implement a code generator producing parallel CUDA code calculating the geometric product of the related algebra. This code can be used to speed up algorithms.

Our approach takes as input the description of an n-dimensional algebra in terms of a metric or signature and calculates a data structure describing the elementary product of all possible combinations of basis blades. This can be seen as a lookup table that is first optimized according to GA simplifications and then used to generate expressions for the individual result multivector components that only depend on the coefficients of the input multivectors to be multiplied. This corresponds to a *table based* compilation process as described in [5]. Finally, these expressions are translated into parallel CUDA code that calculates the result multivector and can be used for efficient calculation of the geometric product.

As a result, we evaluate the performance of the parallelized geometric product to get an estimation on the impact of parallel computing on problems in GA.

2 CONTRIBUTION

We present a code generator which translates the description of an arbitrary geometric algebra with associated signature and metric into CUDA code that implements the geometric product for the specified algebra. This code is a building block that can be used in GA algorithms to calculate the geometric product with the help of a NVIDIA GPU. The code generator is written in Java, allowing the integration into a future versions of the Gaalop [6] optimizer.

Our approach consists of the steps outlined below:

1. Input description of signature and metric for the geometric algebra to be optimized.

2. Computation and optimization of a lookup data structure representing elementary products of basis blades for the given algebra.

3. Calculation of expressions for each component of the result multivector.

4. Code generation for optimized output code implementing the geometric product.

2.1 Specification of Algebra

We support arbitrary geometric products to be optimized by our compiler. Therefore, it must be possible to specify an arbitrary n-dimensional geometric algebra by means of a signature and metric description. From this information we derive a $n \times n$ matrix describing the geometric product of basis vectors with $e_{ij} = e_i e_j$.

2.2 Computation of Data Structure

Using the information from the previous step, we calculate a lookup data structure that describes the geometric product of all possible basis blades that can be constructed over the n basis vectors from the algebra. This data structure has the form of a matrix with $2^n \times 2^n$ entries. Note that an arbitrary geometric algebra is non-euclidean, i.e. its metric is not diagonal. So, in general, a single entry of the lookup matrix consists not only of a simple product of basis vectors but rather of a sum of expressions. In the 5D conformal algebra, for example, $e_0 e_\infty = -1 + e_0 \wedge e_\infty$. The maximum number of summands in this algebra is 2, namely the scalar -1 and the 2-blade $e_0 \wedge e_\infty$.

Each entry in the lookup data structure can be viewed as a list of references to basis blades named E_1 to E_N with $N = 2^n$. For a three-dimensional algebra, for example, the basis blades are named as shown in the following table.

E_1	1
E_2	e_1
E_3	e_2
E_4	e_3
E_5	$e_1 \wedge e_2$
E_6	$e_1 \wedge e_3$
E_7	$e_2 \wedge e_3$
E_8	$e_1 \wedge e_2 \wedge e_3$

Table 1: Naming of basis blades.

The internal representation can be made very compact using a bitwise encoding for "active" basis vectors in a blade, e.g. 101 for $E_6 = e1 \wedge e3$ as used in Gaigen [4], for instance. We assume a canonical ordering of blades, starting with scalars, 1-blades, 2-blades and so on, labeled with $E_1, E_2, ...E_N$.

Exploiting geometric algebra properties such as anticommutativity ($e_i \wedge e_j = -e_j \wedge e_i$) and the knowledge of the inner product of basis vectors from the first step, it is possible to simplify the entries in the lookup matrix. The goal is to have only very few references to signed (+/-) basic blades. This applies to euclidean algebras, for instance, where the metric is diagonal and each entry consists of a single blade. The following table gives an example for the optimized lookup table of a 2-dimensional euclidean algebra. More details about the table-based approach of implementing and optimizing geometric algebra operations like the geometric product can be found in [5].

	b	E_1	E_2	E_3	E_4
a		1	e_1	e_2	e_{12}
E_1	1	E_1	E_2	E_3	E_4
E_2	e_1	E_2	E_1	E_4	E_3
E_3	e_2	E_3	-E_4	E_1	-E_2
E_4	e_{12}	E_4	-E_3	E_2	-E_1

Table 2: Example of a lookup table.

We calculate the lookup table using the freely available reference implementation of Gaigen [3], which is implemented in Java and provides basic functionality without performance considerations but serves our purposes. After the calculation of the lookup table, we get a list of basis blades for each entry of the table containing only non-zero references to the basis blades E_i. Each entry represents a part of the expression that contributes to the coefficient of the result multivector. For each reference to basis blades E_i in the entry, the input coefficients associated with the row and column of the position in the lookup table contribute to the final coefficient of blade E_i in the result multivector.

2.3 Multivector Components

From the information stored in the lookup table, expressions for each component of the result multivector can be determined. Each multivector consists of a combination of 2^n basis blades and associated coefficients, where the i-th coefficient is multiplied by E_i. Since only coefficients of the input multivectors are of interest, the references to basis blades found in the lookup table have to be translated. For each multivector component i, the references to the E_i are looked up in the table.

Let a, b be the input multivectors with coefficients $a_1, a_2, \ldots a_N$ and $b_1, b_2, \ldots b_N$. Then for an entry E_i in the lookup table the coefficients are selected according to the column and row where the entry has been found. So, for the first occurrence of E_1 in the example from Table 2, which is placed in the first column and first row, indices a_1 and b_1 have to be selected. We will associate the row index with multivector a and the column index with multivector b, as indicated by Table 2. The sign to be used corresponds directly to the sign of the reference to the basis blade found in the table.

The final expressions for the coefficients of the result multivector c from the above example are shown below.

$$c_1 = a_1b_1 + a_2b_2 + a_3b_3 - a_4b_4$$

$$c_2 = a_1b_2 + a_2b_1 - a_3b_4 + a_4b_3$$

$$c_3 = a_1b_3 + a_2b_4 + a_3b_1 - a_4b_2$$

$$c_4 = a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1.$$

Note that these expressions only consist of a sum of products of coefficients. The GPU which will finally calculate these coefficients and therefore does not require any knowledge about geometric algebra operations like outer or inner product, for example.

2.4 Code generation

The last step in the code generation process is creating the CUDA output. We parallelize the calculation of each multivector component, so each component is calculated in a separate thread. This is possible since each thread only depends on the input coefficients of multivectors a and b. For an n-dimensional algebra there will be 2^n threads executed in parallel. In practice, the actual number of parallel threads depends on the amount of processing units on the target platform.

In fact, the parallelization scheme is application-dependent. For a single application calculating a single geometric product at a time, the above principle is sufficient. But for applications which might have a large number of geometric products to be calculated in parallel at the same time, this principle might be inappropriate. In this case, other concepts like streaming, working queues or warp-level parallelization should be more useful. As a first attempt of a GPU-based implementation in this paper, we stick to the first method for parallelization, calculating one coefficient per thread.

Expressions for different multivector components differ as well in the coefficients to be multiplied as in the structure. Depending on the underlying algebra, some multivector components could be always zero while others consist of a large expression with multiple additions and subtractions. In order to distribute parallel threads calculating different components, it is therefore necessary to find a generic representation of such an expression, since parallel threads have to execute the same code[1]. To cope with differences in the length and structure of these expressions, we define a meta data structure which will be used in the CUDA code produced by the code generator. From this data structure, parallel threads can look up which elements of the input multivectors have to be multiplied and added or subtracted to the current result.

The meta data structure consists of two parts. The first one represents a single *summand* in the expression for a multivector component, e.g. a_1b_3. Since these *summands* can also be counted negatively, there is an additional sign field. This structure is modeled as a C struct *Summand* keeping track of the indices of the input multivectors and the sign. To minimize the memory footprint, the code generator generates summand structures only for summands that actually will be used in the calculations. All the summands are finally stored sequentially in an array. A thread calculating the i-th multivector component must know which summands to select from this array. Therefore, the second part of the meta data structure keeps track of which summands correspond to which expression. This is modeled in another C struct *Info* which stores the offset to the relevant summands. This requires the code generator to store summands in the correct order. For each multivector component there is exactly one info object, so each thread selects the info object according to its thread index. The number of summand objects to be involved in the calculation is determined by the current offset and the offset of the next component or the maximum number of summand objects in the case of the last index.

Calculation of the meta data structure can be done before the actual calculation of the geometric product since the related information is static and not dependent on concrete coefficients of the input multivectors. Finally, each thread needs references to

- input multivectors a, b,
- result multivector c,
- array of summands and
- the array of info objects.

From this information each thread selects the info object of the multivector component to be calculated, from which it gets offset and length of the summand objects to involve. Then for each summand object the coefficients of a and b are multiplied according to the index in the summand info and signed accordingly. This temporary result is aggregated until all relevant summand objects have been processed. The final result is written to the result multivector c. The calculation of a multivector coefficient c_i is illustrated in Figure 1.

[1] This is in fact a shortcoming of CUDA-based applications: It is not possible to access the GPU's parallel processing units to be used for different tasks at the same time.

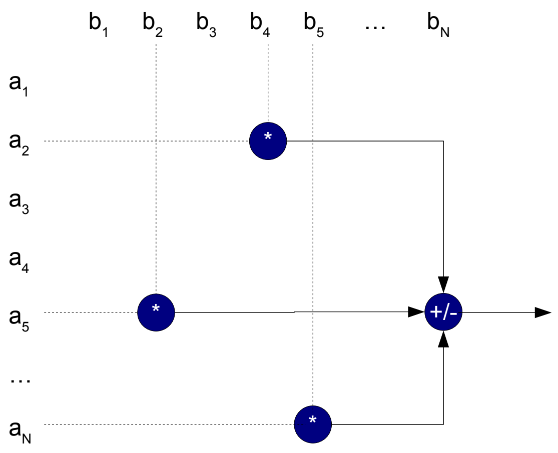


Figure 1: Calculation scheme for coefficient c_i

2.5 Implementation details

When generating CUDA code, it is important to consider the structure of the underlying hardware. The main aspect to be concerned is the memory hierarchy. Depending on the amount of data and the way data is accessed, it is very important to specify which data should be stored in which parts of the device memory. For details on the memory hierarchy, please refer to the CUDA Programming Guide [9].

Listing 1 shows an exemplary CUDA code which is executed by each thread. Variables a and b are vectors containing the input coefficients, c is used to store the results. The offset to the array of summand objects is obtained by the current thread index. According to the index and current offset, the number of summands to be used is determined. Then, for each relevant summand, the indices to the input multivectors are retrieved, and the corresponding coefficients are multiplied and signed. The final result is written into c.

```
__global__ void Calculate(float* a,
                          float* b,
                          float* c) {
  int i = blockDim.x*blockIdx.x+threadIdx.x;
  float res = 0.0;
  int offset = offsets[i];
  int length;
  if (i == N-1) {
    length = num_summands-offset;
  } else {
    length = offsets[i+1]-offset;
  }
  for (int j = 0; j < length; j++)
  {
    Summand s = summands[offset+j];
    if (s.sign) {
      res += a[s.index_a]*b[s.index_b];
    } else {
      res -= a[s.index_a]*b[s.index_b];
    }
  }
  c[i] = res;
}
```

Listing 1: Kernel code executed by each thread

This way it is possible that each kernel executes the same code although the expressions for different multivector components are variable.

The formal parameters to the kernel function, a, b and c, reside in the shared device memory automatically. Since they have to be passed from host side each time a geometric product should be calculated, this is already the place as close to the multiprocessor as possible.

Since the meta data structure consisting of the summand array and the offset objects has to be used in each thread, it is necessary to keep this data in the global device memory. Here, it is important to know that the meta data becomes very large, dependent on the dimension of the algebra. So, for algebras with dimensions larger than 6, other portions of the device memory like shared memory or constant memory are too small. Passing the meta data each time on a kernel call would place the data in the shared memory, which is restricted in size on the one hand and available only for a set of threads from one block on the other hand. Constant memory is cached, so there is in general a performance benefit over global device memory. But constant memory is restricted to 64 KB, which is far too less for large algebras. Details on memory and time consumption will be shown in section 3.

Another problem related to the meta data is the setup procedure, which has to be done before the actual computation can begin. This step is required only once, afterwards an arbitrary number of geometric products can be calculated without the overhead of setting up the data structure again. For each multivector component, there is a variable number of summands to be included in the calculation. Each summand object, which consists of the indices to the input multivectors and the corresponding sign, has to be added to the array of summands. Since in the worst case there are $2^n * 2^n$ summands in total, 2^{2n} objects have to be created and added to the array. This is a considerable amount of code to be produced, because an exponential number of lines of code has to be processed. If the dimension were too high, the produced code would be so large that most compilers would have problems compiling this code. It can therefore be necessary to swap this information out of the program source code, i.e. in a separate file which has to be read at runtime. Doing this leads to a constant size of the actual source code, whereas the size of the input file to be read during the setup increases exponentially with the number of dimensions.

3 EVALUATION

The approach presented in this paper is to be seen as a very first attempt to implement geometric algebra on a multiprocessor platform like GPUs. As a proof of concept, no optimizations have been applied at all. So, allowing to implement the geometric product of an *arbitrary* algebra requires to assume worst case scenarios

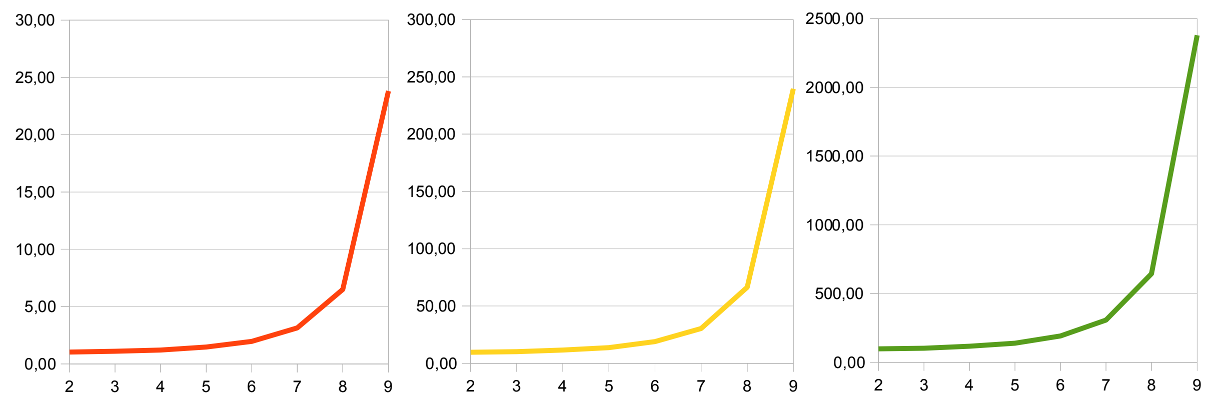


Figure 2: Comparison of time consumption (y-axis) for 10 (left), 100 (middle) and 1000 (right) geometric products. Times are measured in milliseconds. Values on the x-axis specify the dimension of the algebra.

where multivectors all have non-zero coefficients and lookup tables are fully occupied with no entries being zero. Consequently, a euclidean geometric algebra of dimension n is expected to have a lookup table with exactly $2^n * 2^n = 2^{2n}$ elements, each one representing one summand in the different sum-of-product expressions for every coefficient of the result multivector component. Due to this exponential growth and the assumption that multivectors have non-zero coefficients, the meta data required to represent these expressions becomes very large, even for small dimensions.

Performance considerations

Memory

Restricting the maximum dimension of the algebra to 15 allows to encode a summand of some coefficient into a single integer value in the following way[2]:

- 1 bit for sign (e.g. 0 for positive, 1 for negative),
- 15 bits for the index to the first multivector,
- 15 bits for the index to the second multivector and
- 1 unused bit for padding.

Consequently, for an algebra of dimension 15, there would be 2^{15} multivector elements, requiring 2^{15} indices. A summand object, consisting of two indices and the sign would therefore occupy 31 bits of a 32-bit integer variable. As argued above, this would mean to create $2^{2*15} = 2^{30}$ summand objects, each one requiring the space of a single integer, i.e. 4 bytes. The total amount of memory would therefore be $2^{30} * 2^2 = 2^{32}$ bytes = 4 Gbyte.

Obviously, the memory footprint is the limiting factor in this approach. Without optimizations, high dimensions are not feasible at all, because more time would be spent copying the data back and forth from and to the device memory while losing the advantages of parallel processing.

Time

Figure 2 shows three graphs plotting the time to calculate 10, 100 and 1000 geometric products over the dimension of the algebra. For dimensions ≤ 7 with moderate memory footprint, time rises linearly with the number of dimensions. Since memory consumption rises exponentially, higher dimensions have a lot more influence on the computation time, as indicated in the plots. Comparing the plots with respect to the number of geometric products shows that increasing the number of products by factor 10 results in an increase in time consumption by approximately factor 10, too.

These results give a hint that the time required to calculate a number of products is influenced by the memory needed to store the structure of expressions for the different multivector components, which itself depends on the dimension.

Comparison to CPU-based Calculation

The main disadvantage of a GPU-based approach is the necessity to copy data back and forth to and from the device, because the GPU has its own set of memory, registers and caches. Besides copying the meta data required to represent the instructions required to calculate a result coefficient, which has to be done only once per application lifetime, it is necessary to copy the vectors representing the input coefficients onto the device and output coefficients from the device. As opposed to the static meta data, this information changes in each call to the kernel function, so copying has to be performed on each calculation. The effort for copying input and output data is directly related to the dimension of the algebra, due to the fact that each vector of input and output coefficients has 2^n elements, i.e. 2^n times the

[2] Note that otherwise the meta data would be about three times larger, making the application infeasible due to the memory overhead.

size of the data type, e.g. $2^n * 4$ bytes in the case of floats. As an example, calculating the product in an 8-dimensional algebra, $3 * 2^8 *$sizeof(float)$= 3$ KB of input and output data have to be copied for each product. This is a significant amount which has to be considered when measuring the performance of the implementation. In our benchmarks, the time spent for I/O operations between host and device took up to 25% of the overall computation time.

Gaigen [4] is a CPU-based implementation generator for geometric algebra. Gaigen uses advanced optimization techniques and uses specializations that tell the tool where further optimizations can be made, according to the type of multivector. As a CPU-only application, Gaigen does not suffer from memory overhead which is inevitable on parallel platforms as GPUs, for instance. This is why Gaigen is about factor 10 faster than our approach without any optimizations. In turn, Gaigen suffers from a similar problem as outlined in section 2.5, what makes Gaigen unusable for dimensions larger than 6 because the generated code becomes too large so compilers have problems compiling it.

Without considering the memory overhead, our approach is slightly faster than Gaigen, even without applying further optimizations on the structure of multivectors. With a certain knowledge about the types of multivectors which are about to be multiplied, a lot of computation time can be saved by removing parts from the meta data that are actually zero. For example, let the second half of both multivectors be always zero. Then 75% of the lookup table are zero and meta data shrinks to 25% of the worst case, which finally leads to better overall performance. For details, please refer to [5].

Moreover, our approach targets applications using high-dimensional geometric algebras. Only in cases where the dimension is high enough, memory overhead can be compensated by exploiting the parallel architecture, i.e. keeping the device fully loaded.

4 CONCLUSION

We have shown a very first approach of implementing geometric algebra on the GPGPU platform CUDA. We implemented a code generator producing an implementation of the geometric product on the CUDA platform. Applications running CUDA-enabled hardware are therefore able to use this implementation to make use of the computational power of modern GPUs. Without optimizations it is currently theoretically possible to calculate geometric products in algebras of dimension up to 15 without exceeding device memory restrictions[3]. In practice, this is not usable, since there would be too much memory overhead for this number of dimensions.

[3] For dimensions > 10 it is necessary to put input and output multivectors in global memory since shared memory where kernel parameters reside is limited to 16 KB.

Supporting arbitrary algebras while always considering the worst case of full multivectors with non-zero coefficients is too limiting. For most applications, e.g. using the 5D conformal geometric algebra, more than half of the multivector coefficients are usually zero. This is important to know in advance, because multiplication tables as mentioned in section 2.2 can be reduced to a large extent. This is a nice property which has to be exploited as much as possible. Otherwise, there is far too much "infrastructure" required to manage the calculation for different multivector components, as outlined in the previous sections. Since time consumption depends directly on the number of dimensions and the related amount of meta data required for calculation, reducing the lookup data and related memory consumption on the device to the minimum possible value is the most important step to achieve reasonable performance while supporting high-dimensional algebras. Of course, this requires the knowledge of multivectors to be multiplied. One solution is to tell the code generator which specializations of multivectors will be used. This is done in Gaigen, for instance. But creating specializations requires the user to know the structure of multivectors in advance. In complex algorithms, there might be situations where it is hard to decide which parts of multivectors might always be zero. It is therefore desirable to have a tool that optimizes parts of algorithms automatically like Gaalop [6]. This is why the integration of a code generator for parallel platforms into Gaalop is planned for future releases.

5 FUTURE WORK

Our next step will be to investigate the potential for minimization of memory needed to store and process multiplication tables. This can be done automatically by using the algorithm optimizer tool Gaalop, which optimizes algorithms written in the interactive visualization and calculation tool CLUCalc [10] and produces different output formats like C++, for instance. With the help of Gaalop, it will be possible to detect the structure of multivectors and to optimize multiplication tables according to the automatically detected specializations as described in [5].

Gaalop [6] makes it possible to optimize algorithms in GA rather than single calculations. Therefore, the code generator for parallel architectures will be integrated into the Gaalop optimizer software.

Furthermore, we will study existing libraries for geometric computing like described in [1] to find new methods how to generate optimized code for arbitrary algebras and dimensions.

A new standard for parallel computing, OpenCL [7], has recently been released. Using OpenCL makes it possible to address multiple computing devices like CPUs, GPUs or cell processors to be used for general-purpose computing. This standard generalizes vendorspecific GPGPU approaches like CUDA or ATI Stream [2], for example. We see this as an important step for future developments of general-purpose computing. Since as from now OpenCL drivers are officially released by NVIDIA and OpenCL is supported in the latest version 10.6 of the Mac OS operating system, we will extend our code generator to produce OpenCL code in order to support different hardware platforms automatically.

REFERENCES

[1] John Browne. The GrassmannAlgebra Book Home Page. Available at http://grassmannalgebra.info/grassmannalgebra/book/, 2002.

[2] AMD Corp. The ATI Stream Technology home page. http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx, 2009.

[3] Leo Dorst and Daniel Fontijne. Geometric Algebra for Computer Science. http://www.geometricalgebra.net/reference_impl.html, 2009.

[4] Daniel Fontijne. Gaigen 2: A Geometric Algebra Implementation Generator. In *GPCE'06*. ACM, 2006.

[5] Dietmar Hildenbrand. Geometric Algebra Computers. *submitted to the proceedings of the GraVisMa workshop, Plzen*, 2009.

[6] Dietmar Hildenbrand and Joachim Pitt. The Gaalop home page. http://www.gaalop.de, 2008.

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

[8] H. Lange, F. Stock, D. Hildenbrand, and A. Koch. Acceleration and Energy Efficiency of a Geometric Algebra Computation using Reconfigurable Computers and GPUs. *FCCM*, 2009.

[9] NVIDIA. The CUDA home page. http://www.nvidia.com/object/cuda_home.html, 2009.

[10] Christian Perwass. The CLU home page. HTML document http://www.clucalc.info, 2008.

[11] M. T. Pham, K. Tachibana, E. M. S. Hitzer, T. Yoshikawa, and T. Furuhashi. Classification and Clustering of Spatial Patterns with Geometric Algebra. *AGACSE*, 2008.

Creating Editable 3D CAD Models from Point Cloud Slices

Antonis Protopsaltou
University of Ioannina
Department of Computer Science
GR45110 Ioannina, Greece
antonis@cs.uoi.gr

Ioannis Fudos
University of Ioannina
Department of Computer Science
GR45110 Ioannina, Greece
fudos@cs.uoi.gr

ABSTRACT

We introduce a novel approach to reconstructing 3D objects from cross sections of point clouds acquired by laser scanning. Cross sections are almost planar clusters of 3D points. We first thin each cluster to obtain an ordered one dimensional set of points. We then partition the point set to subsets that can be approximated adequately by piecewise quadratic or cubic rational Bezier curves using an optimal fitting method. For each curve we select a number of representative points that lie on the fitting curves which are then used for reconstructing the object surface. Inter-cross section and intra-cross section constraints are imposed to support parameterization and editing of the derived model. Shape and topological differences between adjacent object contours cause severe difficulties in the 3D reconstruction process. By using the contour skeleton information we create intermediate slices representing places where ramifications occur to achieve robust covering (meshing) of adjacent slices.

Keywords

Mesh reconstruction, slicing, thinning, cross sections, reverse engineering, curve fitting.

1. INTRODUCTION

Reverse Engineering is a complex process that is central to industry, arts, archaeology and architecture. In this paper we focus on re-engineering solid objects for which we have acquired the point cloud of their boundary. Subsequently we wish to obtain a 3D CAD model which is editable and manufacturable. Most previous approaches have dealt with this problem considering only mechanical parts and employing feature-based knowledge to detect and represent holes, chamfers, extrusions or protrusions. It is important to provide means for editing 3D objects that respect object morphology and topology.

Various authors have considered creating reverse engineered 3D models. Some researchers have dealt with the tedious task of making their model editable. This is often accomplished by incorporating local and global geometric constraints in the CAD model. In plain solid reconstruction a geometric models is captured directly from the geometry of the point cloud acquired by 3D laser scanning. This method is commonly used in modeling sculptures in arts. These techniques are quite accurate but do not support large scale modifications, additions or other high level operations to the extracted model.

Ko et al. [Ko94] discuss a method that uses a set of points to model a human face. The discussion focuses on the reorganization of the points, facet modeling and tool path generation. Ma and He [Ma98] present an approach to shape a single B-spline surface by a cloud of points. The discussion concentrates on the parameterization of these unorganized points.

Au et. Al [Au99] propose a feature-based reverse engineering method for mannequin in garment design. It is an automated reverse engineering approach for human torsos that creates accurate parameterized models. A key concept in their method is creating a generic mannequin model of a human torso appropriately aligned with the 3D point cloud of the desired human torso model.

Thompson et al [Tho96] have focused on creating high accuracy models of manufactured mechanical parts. The feature based system called REFAB (Reverse Engineering FeAture Based) uses manufacturing features as geometric primitives. The system supports constraints such as parallelism, concentricity, perpendicularity and symmetry depending upon user intervention to extract and accept such features.

Chen and Hoffman [Che95a], define semantics for the creation of generated features. This work is based on a neutral, high-level design representation, called Erep (editable representation), which allows design modifications based on a general design paradigm. This framework considers generated features based on a planar profile and then revolved, swept and extruded in 3D shape.

Dobson et al [Dob95] discuss the fitting of a non-uniform rational B-spline curve to a set of co-planar points. The fitting process uses characteristic points and is demonstrated by fitting a facial 2D profile.

Langbein et al [Lan04] analyze the type of symmetries and shape regularities that may be observed in a Brep model and efficiently apply them in a reverse engineering process to create accurate and aesthetically robust models.

Sato et al [Sat83] propose a laser projection system and an image processor which are used for determining a fixed set of horizontal cross sections of the recognized object which is placed on a turntable in a stable vertical orientation. For each horizontal cross section they compute the Fourier shape descriptors of the boundary. Constraints between two cross sections may be defined such as horizontal strain, section shape, torsion, and displacement.

Werghi et al [Wer99], suggest a general incremental framework whereby constraints may be added and integrated in the model reconstruction process.

Hoppe et al. [Hop92] propose a method for surface fitting based on polygonal meshes. They produce a surface that approximates the original object surface based considering data points in close.

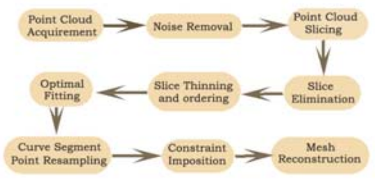


Figure 1: Process Steps

In this paper, we present a novel computer aided reengineering paradigm based on careful slicing of a 3D point cloud and sophisticated post processing of all resulting cross sections. Post processing aims to eliminate noise and partition the point set to point sequences that correspond to low degree curve segments. The curve segments are then approximated using quadratic rational Bezier curves. We then subdivide the curve segments in equilength chord segments and use the corresponding points to perform 3D skeleton-driven mesh reconstruction. Figure 1 illustrates the overall process.

2. CROSS SECTION SELECTION AND PROCESSING

Our reconstruction process starts by slicing the point cloud data, obtained by a 3D laser scanner, into a number of cross sections along a user-specified slicing direction. Most of data points may not be exactly located on a certain cutting plane. Slicing is accomplished by means of a virtual parallelepiped knife with a specified thickness. Slice thickness is controlled by a user defined thickness threshold value that specifies the maximum allowable width of a projected point set. The data points in each slice are projected onto a cutting plane in the middle of the parallelepiped knife perpendicular to the slicing direction. The thickness threshold value is adapted iteratively until it falls under the user specified levels. Slice selection may be controlled by a user defined parameter called slicing distance. Slicing distance specifies the fixed distance between two adjacent slices. There may be cases where the slicing distance is too large for a certain object. As a consequence, the exact geometry of the object is not recorded accurately. The slicing distance parameter should be set according to the object particular features.

In many cases we obtain adjacent slices that are very similar. This might happen when the sliced object feature has symmetric shape such as a cylinder or parallelepiped part. Many of these slices may be eliminated from the entire process of reconstruction. If three adjacent slices are of similar shape, then the middle slice may be eliminated. Similarity of slices may is detected using principal component analysis and skeleton extraction so as to achieve rotational and translational invariability. The analysis of the method is omitted due to space requirements.

Depending on the data acquirement and slicing process a cross-section may contain points that form a shape with thick border. Thinning is the process that identifies the specific points from the data set that are essential to form the actual 2D shape of the cross-section. We call the outcome of this thinning process a thin data set.

The Medial Axis, is a well defined process for extracting a skeleton, but does not always produce a skeleton for the purposes of thinning due to the complexity of the result. Most thinning algorithms available are iterative. The edge pixels are examined against a set of criteria to decide whether they are essential skeleton pixels or not. A common disadvantage of many thinning algorithms is the deformation in the shape of the skeleton at corner and cross regions. Single pixel irregularities may yield gross changes in an otherwise simple skeleton. Furthermore, the extraction of the skeleton does not often preserve the connectivity of the shape. Necking, tailing and spurious projection (line fuzz) are some more defeats of many thinning methods.

The Forced Based thinning algorithm [Par94] is based on the idea that the boundary should be used to locate the skeletal pixels by exerting a force to the inner pixels. In that way, the skeleton of the shape is located at pixels where the forces acting have opposite directions.

All thinning algorithms need as input a 2D array of pixels. In order to convert our unordered set of points to a 2D array of pixels we define a virtual grid of size G_x by G_y where G_x and G_y are the x and y resolution of the grid. Each grid cell would map a certain area in the cross-section. Therefore, every grid cell would contain a number of points that happen to fall in the area specified by

$$(x_{\min}+\frac{x_{\max}-x_{\min}}{G_x}, y_{\min}+\frac{y_{\max}-y_{\min}}{G_y}) \qquad G(\left\lfloor \frac{p_x \cdot G_x}{x_{\max}-x_{\min}} \right\rfloor, \left\lfloor \frac{p_y \cdot G_y}{y_{\max}-y_{\min}} \right\rfloor)$$

where x_{min}, x_{max}, y_{min}, and y_{max} are the minimum and maximum coordinates in the original point set. Consequently, for each point $P(p_x, p_y)$ we add a weight to the grid cell

Each grid cell will play the role of a pixel that is either on or off. We define a grid cell to be on if its weight is greater than the mean weight of all cells in the grid. Figure 2a shows an example grid and the pixels that are on and off.

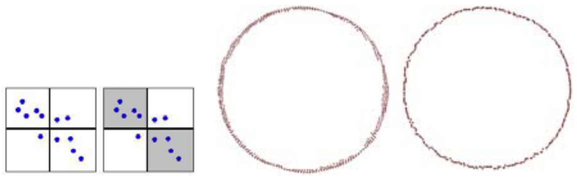

(a) (b)

Figure 2: (a) Virtual grid (b) Thinning

Depending on grid resolution, the above procedure may produce a grid with many not connected pixels. For this kind of cases an anti-aliasing of the grid may be performed in order to fill the gaps between disconnected pixels. The anti-aliasing defines the weight of a cell to be a function of its own weight but also of its 8 neighbors. Each neighbor cell will affect the anti-aliased weight of the cell by its weight multiplied by a coefficient. The following formula computes the anti-aliased weight of a cell:

$$AW_{i,j} = \sum_{m=i-1}^{i+1} \sum_{r=j-1}^{j+1} \frac{W_{m,r}}{\Phi}, \quad \Phi = \left\{ \begin{matrix} 16 & m \neq i & AND & r \neq j \\ 8 & m = i & XOR & r = j \\ 4 & m = i & AND & r = j \end{matrix} \right\}$$

Each grid cell that is characterized on must be mapped with a 2D point. Each grid cell that contains points from the original data set it may be mapped with the centroid of these points. In the case that the cell does not contain any points it may be mapped by the centroid of the points in the 8 neighbor cells. The result may be given as input to the thinning process. Figure 2: (a) Virtual grid (b) Thinning depicts an example of a thick cross-section and the result of quantization and thinning.

3. POINT SET PARTITIONING

In this section we describe the process of detecting the end points of the piecewise Bezier curves.

3.1 Normal Vector Computation

Let two successive points $P_i(x_i,y_i)$ and $P_{i+1}(x_{i+1},y_{i+1})$. The unit normal vector of the line segment that connects the two points is

$$\overrightarrow{U_{i+1}}(\frac{y_i - y_{i+1}}{D_{i+1}}, \frac{x_{i+1} - x_i}{D_{i+1}}), \qquad D_{i+1} = \sqrt{(y_i - y_{i+1})^2 + (x_{i+1} - x_i)^2}$$

D_{i+1} is the magnitude of the N_{i+1} vector. Consequently, each point P_i in the point set P will have two unit normal vectors U_i and U_{i+1} derived from the two adjacent segments $P_{i-1}P_i$, P_iP_{i+1} and therefore we need to calculate the average normal vector which is computed as follows:

$$\overrightarrow{UR_i} = \frac{\overrightarrow{R_i}}{|R_i|} = \frac{\overrightarrow{U_i} + \overrightarrow{U_{i+1}}}{\left|\overrightarrow{U_i} + \overrightarrow{U_{i+1}}\right|}$$

The above equations do not hold for the cases of the first and last points of the point set S where there is only one unit vector component.

3.2 Concavity Change Detection

After computing the unit normal vectors for each point P_i in S we will proceed to the next step of our method which detects inflection points of the final curve. An inflection point is a point on a curve at which the curvature changes sign which actually means that the curve changes from being convex to concave or vice versa.

The algorithm is based on the fact that on a concave curve segment the point unit normal vectors UR_i turn clockwise while on a convex curve segment the point unit normal vectors UR_i turn counterclockwise. Finally on a line segment, the point unit normal vector UR_i is constant. To determine the relative rotation of UR_{i+1} with respect to UR_i we use the cross product of the two vectors. The cross product is defined as follows:

$$\overrightarrow{UR_i} \times \overrightarrow{UR_{i+1}} = \det\begin{pmatrix} URx_i & URx_{i+1} \\ URy_i & URy_{i+1} \end{pmatrix} = URx_i \cdot URy_{i+1} - URx_{i+1} \cdot URy_i$$

Then, in case

- $\overrightarrow{UR_i} \times \overrightarrow{UR_{i+1}} > 0$, UR_{i+1} has been rotated counterclockwise with respect to UR_i
- $\overrightarrow{UR_i} \times \overrightarrow{UR_{i+1}} < 0$, UR_{i+1} has been rotated clockwise with respect to UR_i
- $\overrightarrow{UR_i} \times \overrightarrow{UR_{i+1}} \simeq 0$, UR_{i+1}, UR_i are almost collinear either in the same direction or in the opposite.

The algorithm marks each point in S that according to the above is a point of inflection. P_i is always marked as it is the start point of the first Bezier patch B_1. Each successive marked point will be used as an end point of the currently processing Bezier patch (B_i) and also as the start point of the next Bezier patch (B_{i+1}). Furthermore, the algorithm approximates with a single quadratic rational Bezier patch all points in S whose unit normal vectors form at most $\pi/4$ angle with the unit normal vector of the start point. The angle between two vectors is computed using the dot product: $A \bullet B = |A||B|\cos\theta$ or $A \bullet B = \cos\theta$ for unit vectors.

The algorithm makes use of a tree structure. Aim of the tree is to assist in partitioning the point set in as many partitions as the number of curves that are needed to approximate the point set. The end of a partition would imply a change of curve. Each node of the tree represents a point P_i along with all related information. A tree node may have at most three children *CW* for Clockwise, *CCW* for Counterclockwise, *CL* for Collinear. The tree must grow in the *CW* direction when the point's normal vectors change clockwise, or in the *CCW* direction when the point's normal vectors change counterclockwise, or in the *CL* direction when the point's normal vectors do not change. Therefore a node where there is a direction change represents a point of inflection - Bezier end point.

In many cases the point set may be very noisy. As a result, the direction of the normal vectors changes very frequently and locally thus is creating an abundance of curve patches. This problem could be overcome by performing a data smoothing on the point set prior to using our method but this would increase the time complexity of the entire method. To this effect, we use an alternative that achieves better results by smoothing the normal vectors rather than the actual point set. This smoothing is succeeded by computing the normal vector of each point by averaging the normals of a larger number of neighbour line segments

The algorithm divides the ordered set of points into partitions. All points in every partition preserve the following relation: $P_i < P_k$ for $i < k$ which means that on an upward concaving partition: $UR_1 \bullet URi < UR_1 \bullet UR_k$ for $i<k$, and on a downward concaving partition $UR_1 \bullet URi > UR_1 \bullet UR_k$ for $i < k$.

The above relation is a total order since it preserves

- Antisymmetry: $(P_i < P_k) \wedge (P_k < P_i) \Rightarrow P_i = P_k$
- Transitivity: $(P_i < P_k) \wedge (P_k < P_m) \Rightarrow P_i < P_m$
- Totality: $(P_i < P_k) \vee (P_k < P_i)$

The middle control point is the intersection of the tangent lines to the Bezier curve on the two end points. These tangent lines may be approximated by the lines that pass through the first two points of the partition and the last two points of the partition respectively. Figure 3illustrates this.

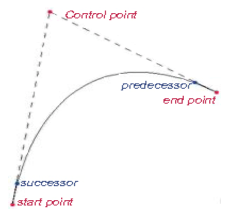


Figure 3: Computation of middle control point

Investigating the efficiency of the algorithm we may easily conclude that the algorithm takes $O(n)$ for an n-point set P each point is processed only once when inserted to the tree structure and only once when traversing the tree to select the inflection points.

4. LOW DEGREE CURVE FITTING

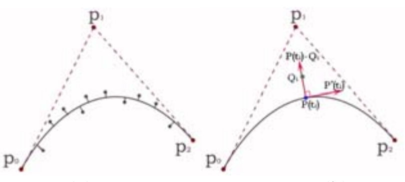
The Bezier representation is one that is utilized most frequently in computer graphics and geometric modelling. Quadratic Bezier curves are often used by CAGD developers since they do not require complex computations as other higher degree curves do. However, in practice it is often desirable to approximate conic sections which cannot be represented in Bezier form. Conic sections such as parabolas hyperbolas and ellipses may be adequately represented by Rational Bezier curves. Non rational Bezier curves are a special case of rational Bezier curves. For these reasons, we will focus on constructing Rational Quadratic Bezier curves. In curve theory, a rational quadratic Bezier curve is defined as follows:

$$P(t) = \frac{\sum_{k=0}^{2} w_k p_k B_k^2(t)}{\sum_{k=0}^{2} w_k B_k^2(t)}, 0 \le t \le 1$$

A 2nd degree Bezier curve requires 3 control points p_k: a start point p_0, an end point p_2, and a 3rd control point p_1 which is obtained by the intersection of the tangent lines on the end points of the curve (see Figure 3).

The B_k terms in the above formula represent the 2nd degree Bernstein polynomials, while the terms w_k are the associated with each control point weights. Setting all weights equal to one the above formula represents an ordinary non rational Bezier curve. Increasing the weight of a control point causes the curve to move towards the specific control point.

The curve fitting process fits equations of approximating curves to the raw field data. Nevertheless, for a given set of data, the fitting curves of a given type are generally not unique. Thus, a curve with a minimal deviation from all data points is desired. For the cases where a rational Bezier curve is approximated the best-fitting curve can be obtained by varying the control point weights (Figure 4a).



(a) (b)

Figure 4: (a) Point distances from the curve. (b) Vectors Q_i $P(t_i)$ and $P'(t_i)$ are perpendicular

A rational Bezier curve *P(t)* that best approximates the given set of 2D points *Q* on a specific cross section is the one that minimizes the sum of the distances of the points from the curve:

$$\sum_{i=1}^{n} \| Q_i - P(t_i) \| = \min$$

Equation 1: Objective function

Also note (Figure 4b) that each vector $\overrightarrow{Q_i P(t_i)}$ is normal to the tangent of the curve at t_i. That means that their inner product is zero. Therefore,

$$\forall Q_i, i = 0 \ldots n, P'(t_i) \bullet (Q_i - P(t_i)) = 0$$

Equation 2: Optimization Constraints

Equation 1 will serve as an objective function while Equation 2 will give *n* constraints. Optimizing the objective function with the constraints may give the weight values of the rational Bezier curve that best fits the given set of points.

5. CURVE SEGMENT POINT RESAMPLING

Point sampling is an important intermediate step for a variety of computer graphics applications. Specialized sampling strategies have been developed to satisfy the requirements of each problem. In this section, we present a sampling technique for 2D models. Our sampling domain is the set of points on a single cross section. Aim of the technique is to generate evenly spaced samples by subdividing the sampling domain into non overlapping parts.

In the previous section we fitted a rational Bezier curve on the points of each cross section. A Rational Bezier curve $P(t)$ is usually defined over the interval *[0, 1]* but it may also be defined over any interval *[0, c]*. The part of the curve that corresponds to *[0, c]* may also be defined by a Bezier polygon. To subdivide the curve [Ran08] to *k* equal length arcs we would first divide the interval *[0, 1]* into *k* subintervals of length *1/k*. The end points of each arc R_i are $P(t_{i-1})$ and $P(t_i)$ where $t_i = i/k$ and $i = 0 \ldots k$. The length of each chord $\| P(t_i) P(t_{i+1}) \|$ converges to the length of the arc R_i between t_i and t_{i+1} when *k* is a rather large value. Consequently, the length of the rational Bezier curve may be approximated by

$$\Lambda = \sum_{i=0}^{k} \| P(t_i) - P(t_{i+1}) \|$$

Equation 3: Bezier Curve length

Considering that the size of the sample set S of points is μ:

$$\forall s_i \in S, i = 0 \ldots \mu - 1, \| s_i s_{i+1} \| = \Lambda / \mu$$

Figure 5: Sample points (in red)

The last relation ensures that all points in the sample set *S* are evenly spaced by a distance of *Λ/μ*. All other points that do not satisfy the above relation are discarded and will not be used in the surface reconstruction process (Figure 5).

6. CONSTRAINTS FOR REDESIGN

The objective of the entire method is to obtain an editable CAD model that would assist us in redesigning the original object. Editability in CAD is commonly achieved by using geometric constraints. When using the term constraint in CAD we usually refer to geometric dimensions and relations (lengths, angles, tangency, parallelism, perpendicularity, etc.) used to define accurately a particular solid geometry. There are two types of constraints that are used in our method.

The set of constraints associated with the geometric properties of the contours [Fud96] in a given cross section will be called Intra-Cross section constraints. Some of these may be point – line segment coincidence, tangency, distance from a curve or point, angle with a curve, parallel – perpendicular line segments or tangents**.**

The second category of constraints is associated with the geometric and topological relationships among the contours of different cross sections which we will call Inter – Cross section constraints. Some of these are point co-linearity, co planar line segments, equality or relation of distances-angles, curve translation, curve scaling.

A system of geometric constraints is then built and the case is treated as a non-linear optimization problem. To solve this system we employ a local non-linear optimization algorithm called Interior Point Optimization. The disadvantage of this method is that it may be trapped in local minima, which makes it depending heavily on the initial configuration. The user is thus advised to make incremental editing. Using global optimization methods or other constraint solving techniques is an interesting research problem [Fud97].

7. MESH RECONSTRUCTION

The task of surface reconstruction deals with the creation of a ribbon between two adjacent cross sections. This may be accomplished by performing triangulation between the sampled sets of vertices that belong to a pair of adjacent cross sections. In most real cases the material of interest lies in the region that separates the adjacent contours.

A rather simple solution that forces a connection of each vertex of a section with some vertices of the adjacent sections was proposed by the literature in the past. However, as the distance between two cross sections may vary, the chance of missing important information of the places where ramifications occur is rather high. As a result, the reconstructed object does not have the correct shape. To overcome this problem, we propose a method that automatically creates intermediate sections.

The projection of the region, which separates the adjacent cross sections, on an intermediate parallel plane is the region that is not common to both contours. We will denote a cross section as a binary image where the two values represent the background and the object. This intermediate plane projection may be expressed as an exclusive OR (XOR) operation on the binary images of the two contours [Chr78]. In cases where the adjacent sections contours intercept, it is required to include the pixels of the contour boundary where the interception occurs.

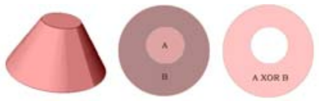
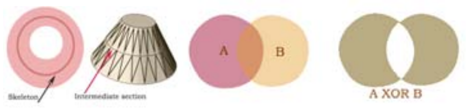


Figure 6: XOR operation on sections A and B

The result of the XOR operation is also a binary image whose boundary is formed by the contours of the contiguous sections. Figure 6 shows that the outer border of the binary image is formed by the second contour while the inner border is formed by the first contour.



(a) (b) (c) (d)

Figure 7: (a) Thinning. (b) Ribbon Construction. (c) Slanted cylinder slices (d) XOR operation

In many cases we may see portions of the binary image to have both inner and outer borders formed by the same contour. This is an indication that in the particular portion of the material of interest there is a ramification. For these cases the skeleton of that portion of the binary image may be used to represent the place where the ramification occurs at an intermediate height of the analyzed sections.

Applying a thinning algorithm on the binary image we may obtain its skeleton (Figure 7a,b). Using the shortest diagonal algorithm [Far97] we are able to create two ribbons (one with each slice).

There are cases though where the XOR operation on the binary images of the contiguous contours does not result in a correct intermediate plane projection. Let's take an example with a slanted cylinder object. As we see in Figure 7c, slice A and B have a small region in common. The XOR operation would derive the region shown in Figure 7d,

Rather than abandoning the method, we will map the contours onto a unit square prior to XOR operation (Figure 8). Each contour is mapped in the following manner:

1. Define a rectangular window which encloses the contour.
2. Calculate $\Delta X, \Delta Y, \overline{X}, \overline{Y}$
3. Map onto a unit square centered at (0,0) by translating and scaling the contour such that its window matches the unit square's window. The equations for this are: $X' = (X - \overline{X}) / \Delta X$, $Y' = (Y - \overline{Y}) / \Delta Y$

Using this mapping we may compute correctly the XOR binary images for slanted and toroidal objects.

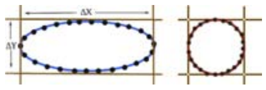


Figure 8: Mapping a contour

The supposition made in the beginning of this section does not cover though all strange cases where the material of interest does not lie in the region that separates the two adjacent cross sections.

8. IMPLEMENTATION AND EXAMPLES

We have implemented and tested the proposed method using the MS Visual C++ programming language, the OpenGL graphics libraries, and the IpOpt optimization software [Thea] and the ACIS by solid modeling libraries Spatial Corporation [Theb]. The implementation was tested on an MS Windows platform. The method was tested for several cloud point sets. Following is an example that illustrates the effectiveness and efficiency of the method.

To demonstrate how this method works we have used a 3D point cloud of a screwdriver object containing 27500 points which was then sliced to equidistant parallel cross sections (**Figure 9**). Figure 10a shows part from a cross section of the screwdriver's handle containing 437 points. Thinning and quantization of this cross section results in a point set with 323 points that form a 1-point-thick curve boundary.

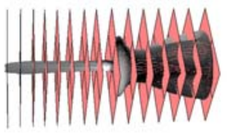


Figure 9: Slicing the screwdriver point cloud

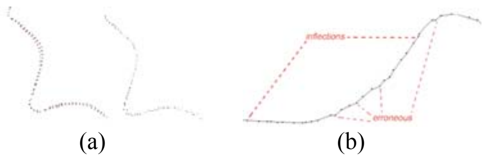


Figure 10: (a) Thinning a cross section. (b) Concavity change detection

While partitioning (Figure 10b) the thin slice point set, the algorithm filters out all noisy points. The final result of the concavity detection process is a set of contours that have the same concavity direction and may be approximated by a low degree rational Bezier curve.

Traversing the point tree, we obtain the end points for each contour that will serve as start and end points for the rational Bezier curves. For the above example, the method processed 323 points and detected 13 inflection points. Figure 11a illustrates the inflection points in blue color. The original point set forms a six peak star shape which is a symmetrical shape. Therefore, we would expect to have 12 quadratic rational Bezier curves specified by 12 symmetrically placed end points (first and last end points coincide). Instead, the method detected 13 points which means that we have 13 curves.

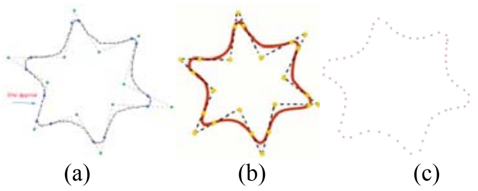


Figure 11: (a) Deriving Control Points. (b) Fitting Rational Bezier Curves. (c) Sampling result.

Figure 11b shows the computation of the middle control point of all quadratic rational Bezier curves that we are going to construct.

To compute the weights for each control point we solve the optimization problem described in section 4 using the IpOpt libraries [Thea]. Figure 11b shows the set of curves built by the algorithm.

Resampling computes the length of each rational Bezier segment in the slice. The approximate length of the entire contour in figure 19b is $\Lambda=95.76$. Setting $\mu=60$, we obtain the distance Λ/μ of each point from its neighbors to be around 1.596. Figure 11c shows the set of representative points that were selected.

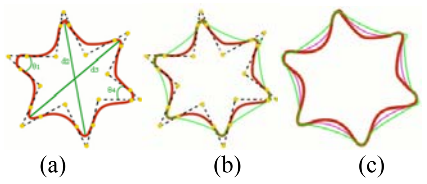


Figure 12: (a) Intra- Constraints: $d_1=d_2=d_3$, $\theta_1=\theta_4$, $\theta_2=\theta_5$, $\theta_3=\theta_6$. (b) Inter Constraints: Slice 1 in red, slice 2 in green. (c) Auto slice generation. Slice 1 in red, slice 2 in green, slice S1 XOR S2 in purple

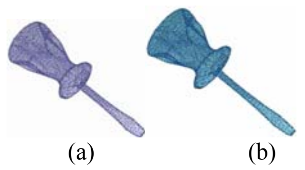


Figure 13:(a) Reconstruction (b) Editing

Our example shows a symmetrical shape of a six peak star. The distance between two opposite peaks is equal for all pairs of peaks. Also the angle formed by the two tangents that intersect on each control point is equal opposite peak angle. These intra–cross section constraints are illustrated in **Figure 12**a. **Figure 12**b shows the two adjacent cross sections where there is a noticeable change in the shape of the six peak star. Despite this, the centroids of the stars are on the same axis. This is actually true for the entire object which makes it a strong inter-cross section constraint. The respective control points are also in the same axis and therefore the difference in the shape is due to the different values of the curve weights. **Figure 12**c shows the intermediate slice generation using the XOR operation. Figure 25a shows the reconstructed object while 25b shows the reconstructed object with its handle's thicker and its shaft longer. To accomplish these edits we increased the diameter of the handle by a factor of 1.3 and increased the distance between the cross sections on the shaft by a factor of 1.4.

9. CONCLUSIONS

We have presented an effective and efficient method to build an editable 3D CAD model from a given point cloud representing the surface of an object. Our method slices the object point cloud into a number of cross sectional points sets that enable us, after fitting

2D curves, to easily impose intra and inter cross sectional constraints on the model that describe accurately the design intent. The constrained model is then reconstructed using triangulation techniques. In future work the reconstruction of the model is done using morphing techniques. Most reverse engineering methods deal with the surface reconstruction problem. Primary aim of our methodology is to create a 3D CAD model that is suited for redesign. The imposed constraints make our model editable. We have evaluated the usability of our method with very good results even for users with no former CAD software experience. Our method provides the tools for robust and accurate editing of the produced CAD model prior to remanufacturing. Automated detection of an optimal slicing direction is an addition that can save users a lot of effort. Finally, the efficiency of the reconstruction process could be improved for complicated objects by first decomposing the object employing convex decomposition methods such as the ones presented in [Bor96] and [Lie06].

10. REFERENCES

[Au99] C.K. Au and M.M.F. Yuen, Feature-Based Reverse Engineering of Mannequin for Garment Design. Computer-Aided Design, 1999. 31, 751-759.

[Bor96] G. Borgefors , G. Sanniti di Baja, Analyzing nonconvex 2D and 3D patterns, Computer Vision and Image Understanding, v.63 n.1, p.145-157, Jan. 1996

[Che95a]X. Chen and C. M. Hoffmann. On Editability of Feature – Based Design. Computer Aided Design, 27(12):905-914, 1995.

[Chr78] H.Christiansen, and T.Sederberg, Conversion of complex contour line definitions into polygonal element mosaics. Computer Graphics 13 1978 187-192

[Dob95] Dobson GT, Waggenspack Jr. WN, Lamousin HJ. Feature based models for anatomical data fitting. Computer Aided Design 1995; 27(2):139–46.

[Far97] G. Farin, Curves and Surfaces for Computer Aided Geometric Design:A Practical Guide, Boston: Academic Press, 1997.

[Fud96] I. Fudos and C. M. Hoffmann - Constraint-Based Parametric Conics for CAD - "Computer Aided Design", Vol. 28, No. 2, pp. 91-100 [01 Jan 1996]

[Fud97] I. Fudos and C. M. Hoffmann - A Graph-constructive Method to Solving systems of Geometric Constraints - ACM Trans. of Graphics, 1997, Vol. 16(2), pp. 179-216

[Hop92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, “Surface Reconstruction from Unorganized Points,” Proceedings of SIGGRAPH 92, pp.71-78, 1992.

[Ko94] Ko H, Kim M-S, Park H-G, Kim S-W. Face sculpturing robot with recognition capability. Computer Aided Design 1994;26(11):814–21.

[Lan04] F.C. Langbein, A.D. Marshall, and R.R. Martin, Choosing Consistent Constraints for Beautification of Reverse Engineered Geometric Models. Comp. Aided Design, 2004. 36: 261-278.

[Lie06] J.M. Lien, J. Keyser, N.M. Amato. Simultaneous Shape Decomposition and Skeletonization, In Proc. ACM Solid and Physical Modeling Symp. (SPM), pp. 219-228, Cardiff, Wales, UK, Jun 2006.

[Ma98] Ma W, He P. B-spline surface local updating with unorganised points. Computer Aided Design 1998;30(11):853–62.

[Par94] J.R. Parker, C. Jennings, D. Molaro. A Force Based Thinning Strategy with Sub-Pixel Precision. In Proceeding of Vision Interface 94 (Banff, AB, 18-20 May 1994).

[Pin03] J.Pina, and R Alquezar, Reconstruction of Surfaces from Cross Sections Using Skeleton Information, CIARP 2003, 180-187

[Ran08] M.Randrianarivony, Arc Length of Rational Bezier Curves and Use for CAD Reparametrization. World Academy of Science Engineering Technology 34, Oct 2008, ISSN 2070-3740

[Sat83] Y. Sato and I. Honda. Pseudodistance measures for recognition of curved objects. IEEE Trans. Pattern Anal. Machine Intell. PAMI-5, 4 (July 1983), 362-373

[Thea] The Ipopt - Interior Point Optimizer Project. https://projects.coin-or.org/Ipopt

[Theb] The 3D ACIS Modeler. ACIS Corporation. http://www.spatial.com

[Tho96] W.B. Thompson, H. De St. Germain, T.C. Henderson, and J.C. Owen. Constructing High-Precision Geometric Models from Sensed Position Data. in In Proceedings 1996 ARPA Image Understanding Workshop. 1996.

[Til83] W. Tiller. Rational B-splines for curve and surface representation. IEEE Comput Graph Appln 1983;3(6):61–9.

[Wan06] W.Wang, H.Pottmann, and Y.Liu, Fitting B-spline curves to point clouds by curvature-based squared distance minimization. ACM Trans. Graph. 25, 2 (Apr. 2006), 214-238.

[Wer99] N. Werghi, R. Fisher, C. Robertson and A. Ashbrook. Object Reconstruction by Incorporating Geometric Constraints in Reverse Engineering. Computer Aided Design, 31(6): 363-399,19

Shading of Bézier Patches

Jan Bocek
University of Ostrava
30 dubna 22
701 03 Ostrava, Czech Republic
jan.bocek@osu.cz

Alexej Kolcun
IG AS CR
Studentská 1768
702 00 Ostrava, Czech Republic
alexej.kolcun@ugn.cas.cz

ABSTRACT

The comparison of Phong shading method and exact shading method for Bézier patches is presented. It is shown that even in bilinear case these shading methods can differ substantially, while the computational complexity is the same. Similar result is presented for quadratic Bézier triangle patches.

Keywords

Beziér patch, shading, normal vector surface.

1. INTRODUCTION

There are well known shading algorithms ([Gour], [Phong]) for the surface shading. The problem is that such algorithms just approximate the real shading.

Precise shading algorithms require the values of normal vectors at all points of rendered surface. Computation of normal vector for Cartesian and triangular parametric surfaces is presented in the paper.

Section 2 describes the computation of the normal vectors in Cartesian case according to the [Yam]. Moreover relation among the "control normals" is formulated and proved. In Sec. 3 we present results for normal patch for triangular Bézier patches.

The comparison of Phong shading and our approach based on direct computation of normals is presented in Sec. 4 for both Cartesian and triangular patches.

The discussion about the patches for which the precise shading is the same as the Phong one, is presented in Sec.5.

2. NORMAL SURFACES

Let the patch $P(s,t)$ of degree (n,m) is analyzed,

$$P(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} b_i^n(t) b_j^m(s) P_{ij}. \qquad (1)$$

Normal vector at a point of a patch can be described as follows

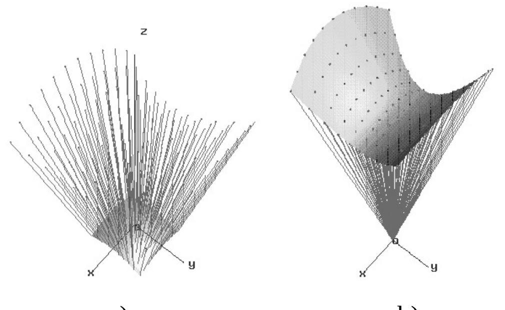
$$N(s,t) = \frac{\partial P(s,t)}{\partial s} \times \frac{\partial P(s,t)}{\partial t}. \qquad (2)$$

For partial derivatives of Bézier patch the formulae below are valid

$$\frac{\partial P(s,t)}{\partial s} = \sum_{i=0}^{n}\sum_{j=0}^{m-1} b_i^n(t) b_j^{m-1}(s) m (P_{i,j+1} - P_{i,j})$$

$$\frac{\partial P(s,t)}{\partial t} = \sum_{i=0}^{n-1}\sum_{j=0}^{m} b_i^{n-1}(t) b_j^m(s) n (P_{i+1,j} - P_{i,j})$$

Resulting normal vectors of (1) we express as a polynomial patch of degree (*2n*-1,*2m*-1).



a) b)

Fig. 1. [Yam]

Figure 1a) shows a patch and its normal vectors at points. Moving these points to the origin gives a normal vector patch – Figure 1b).

$$N(s,t)=\sum_{i=0}^{2n-1}\sum_{j=0}^{2m-1} p_i^{2n-1}(t)\,p_j^{2m-1}(s)\,N_{ij}$$

We can construct the resulting formula in explicit Bézier form:

$$N(s,t)=\sum_{i=0}^{2n-1}\sum_{j=0}^{2m-1} b_i^{2n-1}(t)\,b_j^{2m-1}(s)\,N_{ij} \qquad (3)$$

$$N_{ij}=\sum_{q=B}^{A}\sum_{r=D}^{C}\binom{n}{r}\binom{n-1}{i-r}\binom{m}{q}\binom{m-1}{j-q}\frac{nmQ_{ijrq}}{\binom{2n-1}{i}\binom{2m-1}{j}}$$

$$\begin{aligned} A &= \min(j,m)\\ B &= \max(j-m+1,0)\\ C &= \min(i,n)\\ D &= \max(i-n+1,0)\end{aligned}$$

$$Q_{ijrq}=(P_{r,j-q+1}-P_{r,j-q})\times\left(P_{i-r+1,q}-P_{i-r,q}\right)$$

Formula

$$\sum_{i=0}^{n}\sum_{j=0}^{n-1} t^i a_i t^j b_j=\sum_{k=0}^{2n-1} t^k \sum_{q=\max(k-n+1,0)}^{\min(k,n)} a_q b_{k-q}$$

was used for derivation of (3).

The difference between our and [Yam] approach is that our formula is in explicit form which is better for implementation.

Observing properties of Bézier normal vector patch of Cartesian Bézier patch we can formulate following lemma:

Lemma 1: The control normal vectors N_{ij} fulfil the equation

$$\sum_{i=0}^{2n-1}\sum_{j=0}^{2m-1}\binom{2n-1}{i}\binom{2m-1}{j}(-1)^{i+j}N_{ij}=0\,. \qquad (4)$$

Proof:

We can express (1) as:

$$\begin{aligned} P(s,t) &= (1,t\ldots t^n)\cdot B\cdot P\cdot B^T\left(1,s\ldots s^m\right)^T= \\ &= (1,t\ldots t^n)\cdot Q\left(1,s\ldots s^m\right)^T\end{aligned}$$

which is equal to

$$P(s,t)=\sum_{i=0}^{n}\sum_{j=0}^{m} t^i s^j Q_{ij}^{nm} \qquad (5)$$

where

$$Q_{ij}^{nm}=\sum_{k=0}^{i}\sum_{l=0}^{j}\binom{n}{i}\binom{i}{k}\binom{m}{j}\binom{j}{l}(-1)^{i+j+k+l}P_{kl}\,. \qquad (6)$$

For the case of normal patch of degree (2*n*-1,2*m*-1) the formula (5) is:

$$N(s,t)=\sum_{i=0}^{2n-1}\sum_{j=0}^{2m-1} t^i s^j Q_{ij}^{2n-1,2m-1}$$

where

$$Q_{ij}^{2n-1,2m-1}=\sum_{k=0}^{i}\sum_{l=0}^{j}\binom{2n-1}{i}\binom{i}{k}\binom{2m-1}{j}\binom{j}{l}(-1)^{i+j+k+l}N_{kl}\,.$$

Let us analyze the member with the highest degree

$$Q_{2n-1,2m-1}^{2n-1,2m-1}\,.$$

According to (6) we obtain

$$Q_{2n-1,2m-1}^{2n-1,2m-1}=\sum_{k=0}^{2n-1}\sum_{l=0}^{2m-1}\binom{2n-1}{k}\binom{2m-1}{l}(-1)^{k+l}N_{kl} \qquad (7)$$

Comparing (4) and (7), we can see that we have to prove that

$$Q_{2n-1,2m-1}^{2n-1,2m-1}=0 \qquad (8)$$

is valid.

Let us express (5) in different way separating polynomial member of the highest degree:

$$P(s,t)=t^n s^m Q_{nm}^{nm}+R(s,t)$$

Partial derivatives can be expressed as

$$u=\frac{\partial P(s,t)}{\partial s}=mt^n s^{m-1}Q_{nm}^{nm}+\frac{\partial R(s,t)}{\partial s}$$

$$v=\frac{\partial P(s,t)}{\partial t}=nt^{n-1}s^m Q_{nm}^{nm}+\frac{\partial R(s,t)}{\partial t}$$

$$\begin{aligned} u \times v &= mnt^{2n-1} s^{2m-1} Q_{nm}^{nm} \times Q_{nm}^{nm} + \\ &+ mt^{n} s^{m-1} Q_{nm}^{nm} \times \frac{\partial R(s,t)}{\partial t} + \\ &+ nt^{n-1} s^{m} Q_{nm}^{nm} \frac{\partial R(s,t)}{\partial s} \times + \\ &+ \frac{\partial R(s,t)}{\partial t} \times \frac{\partial R(s,t)}{\partial s} \end{aligned}$$

We can see that the member with the highest degree in resulting cross product of these derivations is equal to zero, because

$$Q_{nm}^{nm} \times Q_{nm}^{nm} = 0$$

QED.

3. CARTESIAN SURFACES

According to the formula (3) we can see that degree of Cartesian patch changes from (n, m) to $(2n - 1, 2m - 1)$. It means that the normal patch for bilinear patch is also bilinear one. Using (4) we can express normal patch for bilinear one as a plane.

This means that the normal vectors of bilinear surface may be constructed as a linear interpolation of normal vectors in control points.

The difference between our direct shading and Phong shading is:

- Phong shading method uses normalized normal vectors in the corners of the patch, and interpolates their values,
- we use directly computed unnormalized normal vectors in all points of the patch.

We, of course, always normalize vectors in both cases.

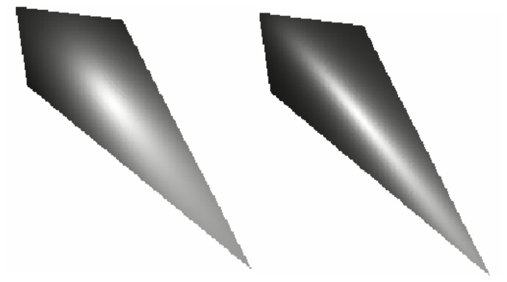


Fig. 2 Shading of the patch.

a) Phong's approach, b) our approach(exact shading).

We have shown that computation complexity of shading is the same for both our and Phong's approaches in case of Cartesian bilinear surfaces.

The results show, that Phong's approximation of shading differs significantly from direct (real) shading. The above mentioned leads to the following:

Lemma 2: Phong shading method is equal to direct (real) shading for the Cartesian bilinear surface if the sizes of all normal vectors in corners are equal.

The example of such surface is shown in Fig. 3 (bilinear patch inscribed into orthogonal parallelepiped).

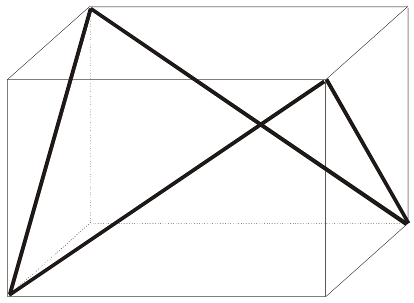


Fig. 3

Following conditions must be fulfilled for general (non-planar) bilinear patch:

$$\alpha + \beta + \gamma + \delta < 360,$$
$$ab \sin \beta = bc \sin \gamma = cd \sin \delta = da \sin \alpha.$$

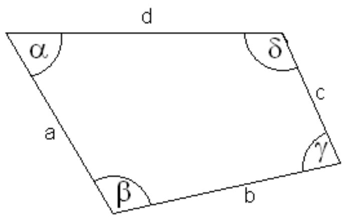


Fig. 4

4. TRIANGULAR SURFACES

Consequently, the idea of the presentation of the normal vector surface for Bézier triangular surface as a Bézier triangular surface is presented here. Given Bézier triangle

$$P(s,t,u) = \sum_{i+j+k=n} P_{ijk} b_{ijk}^{n}(s,t,u) \tag{9}$$

$$b_{ijk}^{n}(s,t,u) = \frac{n!}{i!\,j!\,k!} s^{i} t^{j} u^{k}$$

can be transformed (similar way as in Chapter 2) into polynomial form:

$$P(s,t) = \sum_{k=0}^{n} \sum_{l=0}^{n-k} t^k s^l Q_{kl} \qquad (10)$$

$$Q_{kl} = \sum_{i=0}^{l} \sum_{j=0}^{k} \binom{n}{k} \binom{n-k}{l} \binom{l}{i} \binom{k}{j} (-1)^{i+j+k+l} P_{i+j,i}$$

with $P_{ijk} = P_{i+j,i}$.

Cross product of partial derivations of (10) represents normal surface for triangular surface, and can be expressed as triangular Bézier surface [Boc]. The degree of the resulting normal vector surface is $(2n-2)$. Control points of this surface can be calculated according to the following formula:

$$N_{k+l,l} = \sum_{k_0=A}^{B} \sum_{l_0=C}^{D} E_{kl} T_{k_0+l_0,l_0} \times S_{l+k-l_0-k_0,l-l_0}$$

$$\begin{aligned}
A &= \max(k-n+1,0) \\
B &= \min(k,n-1) \\
C &= \max(l-n+1+k-k_0,0) \\
D &= \min(l,n-1-k_0) \\
E_{kl} &= \frac{\binom{n-1}{k_0}\binom{n-1}{k-k_0}\binom{n-1-k_0}{l_0}\binom{n-1-k+k_0}{l-l_0}}{\binom{2n-2}{k}\binom{2n-2-k}{l}}
\end{aligned}$$

$$T_{ij} = n(P_{i+1,j} - P_{i,j})$$

$$S_{ij} = n(P_{i+1,j+1} - P_{i,j})$$

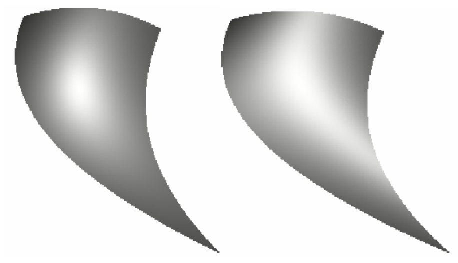
From the point of view of effective visualization, quadratic Bézier triangles seem to be interesting because

- they are natural generalization of planar triangles,
- they are able to model smooth surfaces,
- normal vector surface is also quadratic.

As the quadratic Bézier triangle is the simplest generalization of (planar) triangle, this approach generalizes Phong shading model.

Similarly as in the Cartesian case, the differences between our direct shading and Phong shading are:

- Phong shading method uses normalized normal vectors in the corners of the patch, and then interpolates linearly.
- We use directly computed unnormalized normal vectors in all points of the patch. This approach we can interpret as a quadratic interpolation.



a) b)

Fig. 5 Shading of the patch

a) Phong's approach, b) exact shading.

5. CONCLUSION

We presented Bézier expression of normal surface in explicit form for both Cartesian and barycentric case. We have shown that for the case of bilinear patches and quadratic triangular patches the computation complexity for exact shading based on this approach is comparable to Phong shading method.
Our direct shading can be used as a tool for the difference analysis of shading models.

6. ACKNOWLEDGMENTS

The paper is supported by grant AVOZ 30860518 and by internal grant of University of Ostrava.

REFERENCES

[Boc] Bocek, J., Effective visualization of Bézier surfaces, MSc. Thesis, University of Ostrava, 2007. (in Czech)

[Jin] Jin, S., Lewis, R.,R., West, D.: A comparison of Algorithms for Vertex Normal Computation. The Visual Computer Vol. 21, No.1-2, 2005 pp.71-82.

[Yam] Yamaguchi, Y. Bézier normal vector surface and its applications, SMA '97, p.26, 1997.

[Gour] Gouraud, H., Continuous shading of curved surfaces, IEEE Transactions on Computers, C20(6):623–629, 1971.

[Phong] Phong, B.T., Illumination for computer generated images, Comuniactions with the ACM, 1975

Motion Tracking with Geometric Algebra-valued Particle Filter

Kanta Tachibana

Faculty of Informatics,
Kogakuin University
1-24-2 Nishi-Shinjuku,
163-8677, Shinjuku-ku, Tokyo

kanta@cc.kogakuin.ac.jp

ABSTRACT

Choice of latent variables and likelihood function are important for three-dimensional time-series tracking. I propose motion tracking with high order geometric algebra-valued particle filter. The proposed method shows significant improvement compared to tracking with conventional Euler angles.

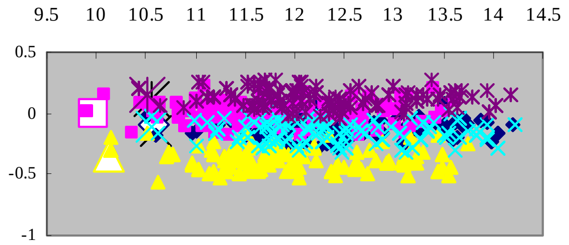
Keywords
Geometric Algebra, Motion Tracking, Computer Vision

1. INTRODUCTION

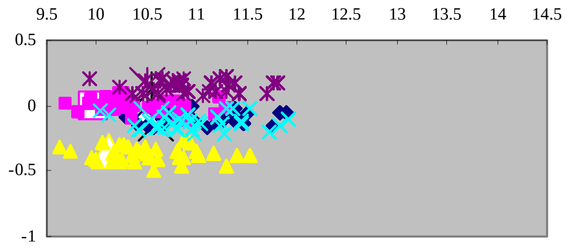
Inference of three-dimensional motion is an important task for computer vision. In this study, I discuss inference of rigid body motion when some feature points are observed by a monocular camera. I assume that feature points are observed at every timestep without occlusion but with measurement noise. For this task, I utilize particle filter (PF), a sequential Bayesian method. For PF to be applied to geometric problems, it is important to use appropriate latent variables. Some studies [e.g. Mar01] utilize quaternion instead of conventional Euler angels as latent variables for time-series three-dimensional inference. However, quaternion's effect has not been clarified quantitatively yet. I furthermore introduce a new resampling step, in which hypotheses are evaluated, for particle filter. In the new resampling step, likelihood of hypothesis is calculated using not only single points but also circles each of which is a combination of three points. The aim of this study is to clarify effects of 1) representations of rotation with rotor components to such geometric inference and of 2) introducing high-order entities to resampling step.

2. NUMERICAL EXPERIMENT

Five feature points fixed at a transparent rigid body, which moves smoothly in 3D space, are assumed to be observed through 100 timesteps. PFs track these feature points. One PF uses Euler angles as its latent variables and the other uses rotor components. The latter evaluates likelihood of each hypothesis under distance from hypothetical circles as well as hypothetical points. Figure 1 shows result of Euler angle PF (a) and that of my proposal (b). Inference of depth (shown as horizontal axis) improved. Average errors were (a) 0.89 (+/- 0.16) vs. (b) 0.59 (+/- 0.05).



(a) Euler angles



(b) Rotor components

Figure 1. True (big marks) and inferred positions (small marks) projected orthogonally into Depth-Height plane.

3. REFERENCES

[Mar01] Marins, J. L. et al. An extended Kalman filter for quaternion-based orientation estimation using MARG sensors. In Proc. IEEE-RSJ Intl Conf. on Intelligent Robots and Syst., pp.2003-2011, 2001

Numerical Method for Accelerated Calculation of Point Light Source Optical Field

Pavel Zemcik
Faculty of Information Technology
Brno University of Technology
Bozetechova 2
CZ 612 66, Brno, Czech Republic
zemcik@fit.vutbr.cz

Ivo Hanak
Faculty of Information Technology
Brno University of Technology
Bozetechova 2
CZ 612 66, Brno, Czech Republic
hanak@fit.vutbr.cz

ABSTRACT
This contribution describes a method for calculation of an optical field generated by a scene illuminated by a monochromatic coherent point light source. The scene optical field calculation is the key part of hologram synthesis methods. Contemporary hologram synthesis methods are extremely computationally extensive and many of them use scene represented through point cloud whose optical field is then rendered using superposition of point light sources. In a previous work by the authors, an approximation has been introduced that uses fixed point type most of the time and does not impose distance limitations. Speedup gain with no loss of flexibility was achieved. In the presented contribution, it is shown that the approximation can be simplified even further and the resulting error does not impose any additional restrictions and can lead in an additional speedup. Furthermore, the numerical aspects of the new approach are shown and precision and data format considerations are discussed.

Keywords
Spherical wave, optical field, point light source, differential computational scheme

1. INTRODUCTION
Optical field synthesis is a crucial step in hologram synthesis. Even though a hologram can be synthesized directly, the focus of the contribution is on the scene optical field because it can be used for generation of several varieties of holograms. In fact, the only necessary step in between the scene optical field and hologram is simulation of interference of the scene optical field and optical field of an appropriate coherent light source known as reference light source.

This contribution focuses on geometrical aspects of the problem of the optical field calculation with an aim to calculate the one sample wide columns of optical field of a point light source (PLS). Such calculation is a time-critical part of some optical field calculation methods and as the optical field synthesis methods on contemporary computers take typically several hours or even days to finish their calculation, any speedup in the implementation is worth the effort.

The target platforms include not only traditional computers, but also programmable hardware, such as programmable gate arrays (FPGA) [Bro96] where the proposed approach goes beyond the known solutions [Ito05] [Nis05] [Mas06]. The proposed approach can also be used in optical field computations through general purpose computation on graphical processing units (GP-GPUs), or some other suitable platforms.

2. OPTICAL FIELD CALCULATION
In the case concerned, the optical field is represented through a uniform square grid of optical field samples $u_{m,n}$, $m \in \lfloor -M,\ M-1 \rfloor$, $n \in \lfloor -N,\ N-1 \rfloor$ on a regular square grid on a plane $\kappa : z = 0$. Each sample $u_{m,n}$ is a complex number whose location in the space is $\mathbf{x}_{mn} = (m\Delta_x, n\Delta_y, 0)$, where Δ_x, Δ_y are the intervals between the samples (grid step). It is assumed that the scene is illuminated only by coherent monochromatic light. The samples represent amplitude and initial phase which is enough to completely represent the optical field under the given conditions [Goo05].

The scene consists of a set of point light sources (PLS). All sources have location $\mathbf{x}_s = (x_s, y_s, z_s)$, plus amplitude, and initial phase represented by complex number $I_s^{1/2} \exp(j\varphi_s)$, where I_s is intensity and φ_s the initial phase. See Figure 1.

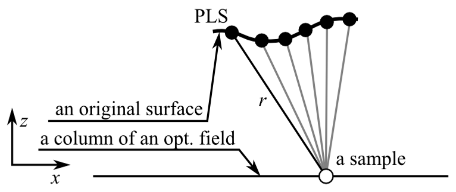


Figure 1: PLS and optical field samples.

The contribution of each of the PLS to the optical field sample is can be evaluated using the Rayleigh-Sommerfeld formula [Goo05], which is often used for this purpose [Ahr06][Ito05][Mas06][Yos00][Nis05]. For simplicity (without loss of generality), it is assumed that no obstacles occur in the scene.

$$u_{m,n} = \sum_s \frac{\sqrt{I_s}}{r} \exp\left(j\frac{2\pi}{\lambda} r\right)\cos\theta, \quad r = \sqrt{(m\Delta_x - x_s)^2 + (n\Delta_y - y_s)^2 + z_s{}^2} \tag{1}$$

where s is a PLS located at $\mathbf{x}_s = (x_s, y_s, z_s)$, θ is the divergence of direction between sample u_{mn} and the PLS s from the plane's normal vector, and λ is the wavelength of the light source.

The contribution of a PLS to the sample u_{mn} in the optical field is, therefore, a complex number, whose phase ϕ_s and amplitude A_s are shown below.

$$\phi_s = \frac{2\pi}{\lambda} r = = \frac{2\pi}{\lambda}\sqrt{(x_s - x_{mn})^2 + (y_s - y_{mn})^2 + z_s^2} \tag{2}$$

$$A_s = \frac{\sqrt{I_s}}{r}\cos(\theta) \tag{3}$$

Figure 2 demonstrates the real part of the values of the optical field on the plane. The projected PLS is, in this case, in the center of the circles close to the right side of the image in Figure 2.

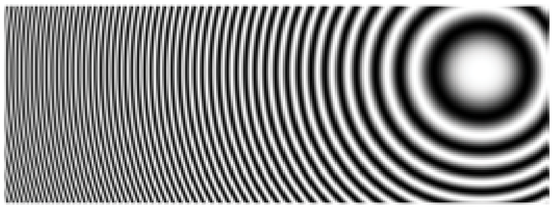


Figure 2: Optical field of a PLS

The above formulas (1)(2)(3) are non-linear and they cannot be approximated directly. However, if the argument of a complex number and the amplitude are separated, as shown above, it is possible to approximate each of the parts using a quadratic function on a small part of a column (hundreds of samples). The approximation can be performed using a differential scheme with fixed point arithmetic as shown in the previous work of the authors [Han09]. The approach is based on the fact that contribution to a piece of a line or column of the optical field can be approximated through quadratic interpolation for the phase and linear interpolation for the amplitude.

Let us consider only a part of a single column of the optical field, for which m is constant and n_i ranges from some n_0 to n_{max-1}, where max is the length of the considered piece of column (max=512 was used in our experiments). The PLS contribution to the optical field based on the amplitude and phase can then be calculated through a component complex number.

$$u_{mn_i} = A_{mn_i}\exp(j\phi_{mn_i}) = = A_{mn_i}\cos(\phi_{mn_i}) + jA_{mn_i}\sin(\phi_{mn_i}) \tag{4}$$

where integer number $n_i \in \lfloor n_0,\ n_{\max-1} \rfloor$.

The sin and cos functions from the equation (4) must also be evaluated. However, for the proposed purpose, the required precision allows for their tabulation in a relatively small table.

The method proposed in this contribution further improves the approximation performance through reduction of complexity of the algorithm.

3. PROPOSED METHOD

In the proposed method, the approximation of the phase and amplitude is being calculated through quadratic and linear scheme.

$$\phi'_{mn} = \phi_A + \phi_B n + \phi_C n^2 \cong \phi_{mn} \tag{5}$$

$$A'_{mn} = A_A + A_B n \cong A_{mn} \tag{6}$$

where ϕ_A, ϕ_B, ϕ_C, A_A, and A_B are coefficients of for quadratic and linear approximations.

The above scheme illustrates the possibility of approximation but in fact, is not suitable for direct implementation as the evaluation of each phase and amplitude is still relatively complex. The scheme can be expressed in a differential form. The starting values for the starting point n_0 are calculated from the initial equations (2) (3) precisely.

$$\begin{aligned} \phi'_{mn+1} &\cong \phi'_{mn} + \Delta\phi'_{mn+1}, \phi'_{mn_0} = \phi_{mn_0} \\ \Delta\phi'_{mn+1} &= \Delta\phi'_{mn} + \Delta\Delta\phi_{mn_0}, \Delta\phi'_{mn_0} = \Delta\phi_{mn_0} \end{aligned} \tag{7}$$

$$A'_{mn+1} \cong A'_{mn} + \Delta A_{mn_0}, A'_{mn_0} = A_{mn} \tag{8}$$

From the above equations (7)(8), it is clear that the approximation step requires two additions for the phase and a single addition for the amplitude. Contribution of a PLS during the approximation is expressed with a pseudocode depicted in Figure 3.

```
$u_{mn} += A'_{mn} \cdot \cos(\phi'_{mn}) + jA'_{mn} \cdot \sin(\phi'_{mn})$
```

Figure 3: Contribution of a PLS to optical field

The step requires three additions, two multiplications, one complex addition, and two table read operations (for sin and cos). It translates into five additions, two multiplications and two table read operations.

The proposed method simplifies this step even further. It is based on the fact that the phase approximation step shown in the above equation (7) can be further simplified without significant loss or precision. The idea of the proposed method is in simplification of the differential scheme. In *k* steps of approximation, the quadratic coefficient $\Delta\Delta\phi_{mn_0}$ is added only once; *k* should be a small integer, *k*=4 was used in our experiments.

$$\begin{aligned}
&\phi''_{mn+1} = \phi''_{mn} + \Delta\phi''_{mn+1}, \phi''_{mn_0} \cong \phi_{mn_0} \\
&\Delta\phi''_{mn+1} = \Delta\phi''_{mn} + \Delta\Delta\phi''_{mn}, \\
(9)\quad &\Delta\phi''_{mn_0} = \Delta\phi_{mn_0} + \tfrac{1}{2}\cdot\Delta\Delta\phi_{mn_0} \\
&\Delta\Delta\phi''_{mn} = \begin{cases} k\cdot\Delta\Delta\phi_{mn_0}, n \bmod k = 0 \\ 0, n \bmod k \neq 0 \end{cases}
\end{aligned}$$

This modification requires that the initial *n* is dividable by *k*, which is not a problem at all for the applications. On the other hand, the modification is quite beneficial due to further simplification – one addition from the above scheme can be avoided.

The removal on a single addition from the scheme is, however, beneficial especially from other point of view. In fact, the quadratic approximation become piecewise linear approximation.

Note, please, that the quadratic part of the equation is corrected for the length *k* as well as the linear part.

The fact that the approximation becomes piecewise linear, of course, affects the precision of the interpolation. However, the precision is only affected within the small linear pieces (see Figure 4). The figure only illustrates the differences between the two approximation and parameters are adjusted to enable their visibility. In reality, the differences are smaller.

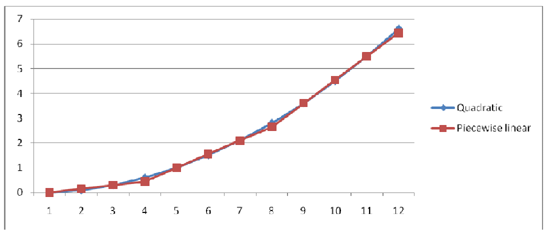


Figure 4: Displacement interpolation in squares

The fact that the approximation is made piecewise linear can further improve the process of calculation of the contribution due to the fact that the table of sin and cos can be modified so that it is not parametrized by the phase ϕ''_{mn}, but also by the difference $\Delta\phi''_{mn}$.

The pseudocode for calculation of the contribution is then treated as one *k*-tuple piece that calculates *k* contributions (see Figure 5).

$$\boxed{\begin{aligned}
&u_{mn} += A'_{mn}\cdot\cos_0(\phi''_{mn},\Delta\phi''_{mn}) \\
&+ jA'_{mn}\cdot\sin_0(\phi''_{mn},\Delta\phi''_{mn}) \\
&u_{mn+1} = A'_{mn}\cdot\cos_1(\phi''_{mn},\Delta\phi''_{mn}) \\
&+ jA'_{mn}\cdot\sin_1(\phi''_{mn},\Delta\phi''_{mn}) \\
&\vdots \\
&u_{mn+k-1} = A'_{mn}\cdot\cos_{k-1}(\phi''_{mn},\Delta\phi''_{mn}) \\
&+ jA'_{mn}\cdot\sin_{k-1}(\phi''_{mn},\Delta\phi''_{mn})
\end{aligned}}$$

Figure 5: *k*-contribution of a PLS to optical field

Note, please, that $\sin_q$ and $\cos_q$ are tables. The format of the tables is such that *q* corresponds to the *q*- th sample of the linear piece. Note also that e.g. in case *k*=4, $\sin_0$, $\sin_1$, $\sin_2$, and $\sin_3$ for the same argument are stored in one memory word so that only a single table access is required for all of them.

While the step above may seem relatively complex, the operation per sample is reduced as shown in the Table 1 below.

Operations per sample	Quadratic (old)	Piecewise linear (new)
Addition	5	4+1/k
Multiplication	2	2
Table access	2	2/*k*

Table 1: Complexity of the approximation steps

Interestingly, the new approach does not require more precise data representation than the previous one. On the contrary, it leads in reduction of the required data width of the $\Delta\Delta\phi''_{mn}$ due to the fact that a larger value is added to an accumulator and the addition is occurring more frequently. The data representation for the other operands remains unchanged.

The data format for the operands is shown in Table 2 below. All the numbers are fixed-point decimal numbers (represented in the computational engine with integer numbers).

Order of bit (2^n)	1	0	-	–1	–2	–3	–4	–5	–6	–7	–8	–9	–10	–11	–12	–13	–14	–15	–16	–17	–18	–19	–20	–21	–22	–23
A'_{mn} (unsigned)	•	•	,	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$\Delta A'_{mn}$ (signed)	←	←	,	←	←	←	←	←	←	←	s	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$\hat{\phi}''_{mn}$ (signed)			,	s	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					
$\Delta\hat{\phi}''_{mn}$ (signed)			,	s	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					
$\Delta\Delta\hat{\phi}''_{mn}$ (signed)			,	s	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					

Table 2: Data representation format for the approximation step calculation

The format of the data shown in Table 2 is different for the amplitude A'_{mn}, $\Delta A'_{mn}$, and for the phase related values ϕ''_{mn}, $\Delta\phi''_{mn}$, and $\Delta\Delta\phi''_{mn}$. The amplitude is positive number so that $A'_{mn} \in \lfloor 0,\ 1 \rfloor$. the $\Delta A'_{mn}$ is a signed number whose value should reflect the difference of the amplitude between the adjacent optical field samples which is always very small so the range is approximately $\lfloor -0.02,\ 0.02 \rfloor$. The representation of the phase related values is based on the periodicity of the sin and cos functions. In fact, the representation is substituted and the substituted values are stored in the computer memory.

$$\hat{\phi}''_{mn} = \phi''_{mn} \tfrac{1}{2\pi}$$

(10) $$\Delta\hat{\phi}''_{mn} = \Delta\phi''_{mn} \tfrac{1}{2\pi}$$

$$\Delta\Delta\hat{\phi}''_{mn} = \Delta\Delta\phi''_{mn} \tfrac{1}{2\pi}$$

This approach allows for storage of only the fractional parts of the phase related values. The integer part can be removed completely as it is never needed. The range becomes $\lfloor -0.5,\ 0.5 \rfloor$ and the sin and cos tables can be easily adjusted to reflect it.

4. CONCLUSIONS

The goal of this contribution was to introduce a novel approach that further simplifies the rendering of optical field of a point light source, which is one of the critical paths synthesis holography calculations.

The method has been presented and it has been demonstrated that it can speed up the calculations. The actual speedup, however, depends on the implementation and the platform. In the presented case, the number of operations per sample was reduced by approximately 25% comparing to the original solution [Han09]. This suggest a speedup of calculation around 20%.

Further work will include in-depth evaluation of the error and its effects on the results and further attempts to speed up the calculations.

5. ACKNOWLEDGEMENTS

The research has been funded by the “Centre of Computer Graphics” (CPG-LC06008), Czech Ministry of Education, Youth, and Sports research grant.

6. REFERENCES

[Ahr06] L. Ahrenberg, P. Benzie, M. Magnor, J. Watson, “Computer generated holography using parallel commodity graphics hardware,” Optics Express, Vol. 14, No. 17, p. 7636—7641, 2006

[Bro96] Brown, S. and Rose, J. FPGA and CPLD architectures: A tutorial. IEEE Design and Test of Computers, pp. 42-57, 1996. IEEE, NY, USA, 1996

[Goo05] J. W. Goodman: Introduction to Fourier Optics, 3rd edition. Roberts & Company Publishers, 2005, ISBN 0-9747077-2-4

[Han09] Hanák, I., Zemčík, P., Žádník, M., Herout, A.: Hologram synthesis accelerated in field programmable gate array by partial quadratic interpolation, In: Optical Engineering, Vol. 8, No. 48, 2009, US, p. 7, ISSN 0091-3286

[Ito05] T. Ito, N. Masuda, K. Yoshimura, A. Shiraki, T. Shimobaba, and T. Sugie.: Special-purpose computer Horn-5 for a real-time electroholography. Opt. Express, 13(6):1923–1932, 2005.

[Mas06] N. Masuda , T. Ito, T. Tanaka, A. Shiraki, T. Sugie, “Computer generated holography using a graphics progressing unit,” Optics Express, Vol. 14, No. 17, p. 7636—7641. 2006

[Nis05] S. Nishi, K. Shiba, K. Mori, S. Nakayama, S. Murashima, “Fast Calculation of Computer-Generated Fresnel Hologram Utilizing Distributed Parallel Processing and Array Operation”, Optical Review Vol. 12, No. 4 (2005) 287-292

[Yos00] H. Yoshikawa , S. Iwase, T. Oneda, “Fast computation of fresnel holograms employing difference,” in Proceedings of SPIE 3956, 48—55 (2000).

Precise Image Resampling Algorithm

Pavel Zemčík
Faculty of Information Technology
Brno University of Technology
Božetěchova 2
CZ 612 66, Brno, Czech Republic
zemcik@fit.vutbr.cz

Bronislav Přibyl
Faculty of Information Technology
Brno University of Technology
Božetěchova 2
CZ 612 66, Brno, Czech Republic
xpriby12@stud.fit.vutbr.cz

Adam Herout
Faculty of Information Technology
Brno University of Technology
Božetěchova 2
CZ 612 66, Brno, Czech Republic
herout@stud.fit.vutbr.cz

ABSTRACT

This paper introduces a precise image resampling algorithm intended for corrections of image distortions caused by lenses or similar devices. The algorithm is designed for correction of small distortions in terms of pixel displacement but with high subpixel precision. The geometrical description of the correction is through bi-linear interpolation within each node of a square or rectangular mesh. The paper describes the algorithms itself, its features, implementation issues and data formats. Specifically discussed are the issues connected with programmable hardware (FPGA) implementation.

Keywords
Image resampling, subpixel resampling, lens distortion, FIR filter, bilinear interpolation.

INTRODUCTION

Image processing is one of the fields of computer science and applications that is developing very fast. The object of image processing is, of course, an image. Vast majority of image processing methods assumes that the image is a 2D signal represented through samples organized in a regular square or rectangular raster [For02a]. While the contemporary image acquisition devices and methods acquire images that relatively well fulfill the above assumption, in most cases, the images suffer from small geometrical imperfections caused e.g. by lenses used with the cameras that acquire the images.

The geometrical imperfections are in some cases not critical; however, many applications of image processing exist that suffer from the imperfections and where it is desirable to correct them. While the geometrical correction – calculation of new sample positions in the image – is relatively straightforward and can be e.g. performed through bi-linear interpolation within square or rectangular mesh, the problem remains how to get the new samples' values so that the signal properties of the image remain as much preserved as possible. Unfortunately, the nearest neighbor method, which completely destroys the image signal properties, and bi-linear or bi-cubic interpolation [Gal05a] which can be better but by far is not ideal, are traditionally used for this purpose. The main reason is that while the algorithms to preserve good signal properties, namely frequency spectrum, are known, they are often considered prohibitively computationally expensive. This paper proposes a method that is far better from the point of view of signal properties than the bi-linear or bi-cubic interpolation while still preserves relatively low computational requirements. The limitation of the proposed method, however, is that it is limited to the cases where the distortions do not involve significant angular or scale changes – the method merely assumes subpixel shift limited to several pixels displacement [For02a], [Gal05a].

IMAGE RESAMPLING

General image resampling problem is relatively straightforward mathematically – it is merely a problem of proper reconstruction of signal values in 2D space and proper application of sampling theorem. However, the efficient implementation of such resampling is still quite open problem. In our approach, we limit the general problem to resampling in order to correct geometrical imperfections only. This limitation has the following implications:

- The displacement of pixel location of the original and resampled images is only units of pixels,
- no angular distortion is expected, and
- no scaling is expected.

The general approach for resampling in our case is to scan the output image raster pixel by pixel (sample by sample) and reconstruct the values from the original raster based on knowledge of the pixel displacement. Taking into account the above limitations, it is known that the sampling theorem cannot be violated and also it can be assumed that the function is separable.

(1) $r_{x,y} = f\left(o, d(x,y)\right) = f''\left(f'\left(o, d_x(x,y)\right), (x,y)\right)$

where ***r*** is the resampled image,

o is the original image,

d is the displacement function,

f is a resampling function,

and ***f'*** and ***f''*** are the partial reconstruction functions after separation.

In our case, the functions ***f'*** and ***f''*** are implemented through a bank of FIR filters indexed by subpixel location of the pixel. Moreover, the sampling is the same in both directions, so ***f'*** and ***f''*** are implemented using the same FIR bank.

The above solution with FIR filters was chosen as it has well defined features and as it is quite flexible in terms of exchangeability of the filtering function.

PROPOSED RESAMPLING

The proposed approach to resampling relies on the separability; however, in addition to the generally used approach, the separability is applied to both the resampling function itself and the geometrical distortion calculation so that the distortion calculation is separated in vertical and horizontal directions.

(2) $r_{x,y} = f''\left(f'\left(o, d_x(x,y)\right), d_y(x,y)\right)$

where **r** is the resampled image,

o is the original image,

$\boldsymbol{d_x}$ and $\boldsymbol{d_y}$ are the displacement functions,

and ***f'*** and ***f''*** are the partial reconstruction functions after separation.

The resampling function itself is assumed to be some suitable filter function and in the presented approach it is implemented through a bank of FIR (Finite Impulse Response) filters [Rab78a]. The bank of FIR filters is indexed through a subpixel position of the desired sample in the raster. The reason is that the FIR coefficients are dependent on the subpixel position of the desired sample location. Of course, the size of FIR filters is limited. The filters in the bank can be e.g. Lanczos filters [Theu00a] for optimal exploitation of the bandwidth of the image signal given the size of the filter, or other filter design to achieve the desired image signal properties. The described approach is, in fact, not dependent on it.

(3) $r_{x,y} = FIR_{fp(x)}\left(FIR_{fp(y)}\left(o, ip\left(d_y(x,y)\right)\right), ip\left(d_x(x,y)\right)\right)$

where ***r*** is the resampled image,

o is the original image,

$\boldsymbol{d_x}$ and $\boldsymbol{d_y}$ are the displacement functions,

$\boldsymbol{FIR_t}$ is the function of the bank for position ***t***,

fp is the fractional part of a numerical value,

and ***ip*** is the integer part of a numerical value.

The distortion to be corrected is described with a square mesh with displacement specified for each node of the mesh. While the displacement in each node (corner of the squares) of that mesh is known, the displacement inside the squares is computed via bi-linear interpolation.

Distortion inside each square is described by means of the following four pre-calculated coefficients:

- D_0 – top left pixel displacement.
- DC_0 – difference of displacements between adjacent pixels in 1st row of the square.
- DR – difference of displacements between 1st pixels of adjacent rows.
- DDC – change in difference of displacements between pixels of adjacent rows, that means $DC_{n+1} – DC_n$.

For more detailed description see Figure 1. Note, please, that the displacement calculation can be subdivided into independent calculation of vertical and horizontal displacements.

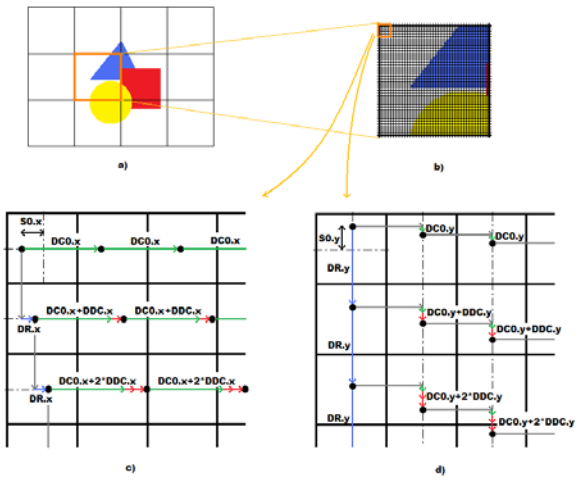


Figure 1: Displacement interpolation in squares

The following pseudocode illustrates displacement calculation resampling algorithm executed in each square of the mesh. The input of the algorithm is the original image, distortion description (through the above mentioned coefficients), and FIR filter; the output is the resampled pixels within the given square. Note, please, that two instances of the algorithm are being used, one for vertical and one for horizontal displacement and filtering.

```
var DoR, DC, D;
DoR = $D_0$;
DC = $DC_0$;
(foreach row in square)
{
  D = DoR;
  (foreach pixel in row)
  {
    Output FIR[fp(D)](O,ip(D));
    D += DC;
  }
  DoR += DR;
  DC += DDC;
}
```

As it can be seen from the pseudocode, three variables are needed in the algorithm. Their meaning is as following:

- DoR – displacement of 1st pixel in a row.
- DC – difference of pixel displacements.
- D – displacement of current pixel.

As shown in the algorithm, the displacement is subdivided into integer and fraction parts. The integer part is used to determine the pixel placement of the filter while the fractional part is used to determine the set of coefficients within the filter bank. When the number of filters in the bank is N (e.g. 16), the fractional part is multiplied by N and then rounded to nearest integer. Then it is used for filter bank index.

FPGA IMPLEMENTATION

An FPGA [Bro96a] implementation of the resampling algorithm has been prepared as part of the experiments with the design. The dataflow in the resampling unit can be seen in Figure 2. The processing contains four parts which are associated in two groups. One group handles vertical resampling while the other handles horizontal resampling. Each group consists of a FIR module and Displacement interpolation module.

Data formats used in the algorithm are the fixed decimal point numbers in order to represent the data accurately enough while maintaining the design simple to enable its simple implementation.

The actual data formats used in the experiments are shown below.

- Pixel data – 16 bit signed or unsigned. The pixel processing is assumed in 16 bit format in order to support the standard dynamic range of contemporary video cameras, which is 10 to 14 bits, plus an overhead for absorption of rounding errors of FIR.
- Co-ordinate – 12+4 bits unsigned. The subpixel resolution is assumed to be 16 subpixel positions which is in practical terms enough to avoid measurable adverse effects of granularity in subpixel position.
- Difference of co-ordinates – 2+8 bits signed. The difference of positions must be precise enough to represent the change of displacement.

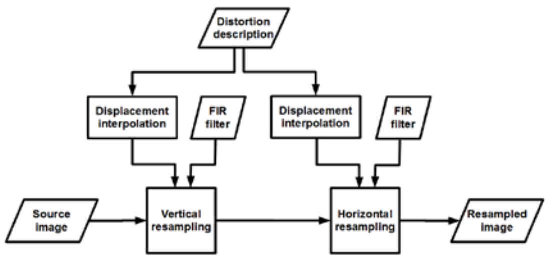


Figure 2: Dataflow of resampling algorithm.

The experimental design and synthesis of the resampling unit was performed for Xilinx Virtex-II xc2v1000 FPGA device. XST version H.38 was chosen for this task. As the unit is relatively generic, the following parameters were used: Image size 256 x 1024 px and square size 64 px which results in square mesh of 4 x 16 squares (and also displacement coefficient sets). Device utilization with configuration mentioned above is shown in Table 1.

Items on chip	Used	Capacity	% capacity
Slices	3 947	5 120	77%
Slice Flip Flops	2 212	10 240	21%
4 input LUTs	3 103	10 240	30%
BRAMs	20	40	50%

Table 1: Exploitation of FPGA unit Virtex II-1000

The device clock frequency is up to 105 MHz. While the resampling unit produces one output pixel per 2 clock cycles, the output resampling data rate for a single unit is up to 52.5 Mpixels per second. This demonstrates the real-time potential of the design.

RESULTS

The algorithm has been evaluated with images of cells obtained through microscopy, synthetic image with various shapes, and other images not shown.

All the images were resampled using a geometrical correction of some lens imperfections. Along with the images themselves, their energy spectrum is shown to demonstrate very little loss of energy caused with the actual resampling. The resampling itself was performed with 7-sample Lanszos filter and the subpixel resolution was 16. See Figure 3 and Figure 4 for the examples.

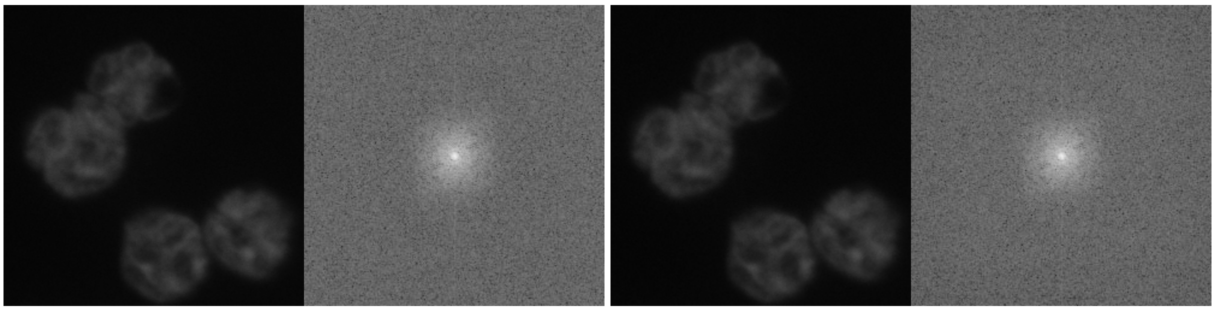


Figure 3: Cells – original image and its spectrum (left), resampled image and its spectrum (right)

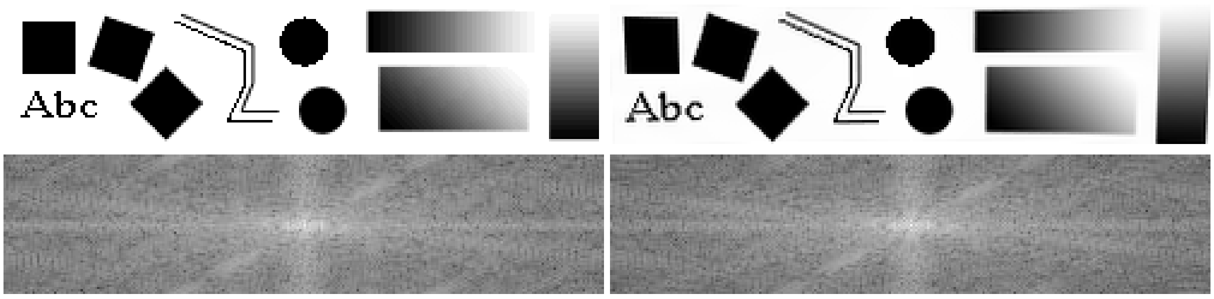


Figure 4: Shapes – original image and its spectrum (left), resample image and its spectrum (right)

CONCLUSIONS

The goal of the contribution was to present a new resampling algorithm that is intended for corrections of geometrical distortions caused during image acquisition.

A new algorithm has been presented that exploits separable FIR filters and also separable displacement calculation for vertical and horizontal directions while it does not adversely affect the image.

The algorithm has been implemented and prepared also for FPGA exploitations. Using the Lanczos filter, the algorithm has also very good results in terms of quality of image signal. Future work should include further simplification of the algorithm.

ACKNOWLEDGMENTS

This work was supported by the grant project of the Ministry of Educatuon, Youth and Sports of CR, (MSMT 2B06052) project “BioMarker”.

REFERENCES

[Bro96a] Brown, S. and Rose, J. FPGA and CPLD architectures: A tutorial. IEEE DESIGN and TEST OF COMPUTERS, pp. 42-57, 1996.

[For02a] Forsyth, D. A. and Ponce, J. Computer vision: A modern approach. Prentice Hall Professional Technical Reference, New Jersey, 2002.

[Gal05a] Gallagher, AC. Detection of linear and cubic interpolation in JPEG compressed images. In proceedings of The 2nd Canadian Conference on Computer and Robot Vision, pp. 65-72, Victoria, BC, Canada, 2005.

[Rab78a] Rabiner, LR and Gold, B. and Yuen, CK. Theory and application of digital signal processing. IEEE Transactions on Systems, Man and Cybernetics, vol. 8, nr. 2, pp. 146, 1978.

[Theu00a] Theußl, T. and Hauser, H. and Gröller, E. Mastering windows: Improving reconstruction. In Proceedings of the 2000 IEEE symposium on Volume visualization, pp. 101-108, ACM New York, NY, USA, 2000.

Context based controlled Virtual Tours using Viewpoint Entropy for Virtual Environments

RNDr. Ján Lacko
lacko@sccg.sk
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava
Mlynská dolina
842 48 Bratislava, Slovakia

Marian Maričák
marianmaricak@gmail.com
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava
Mlynská dolina
842 48 Bratislava, Slovakia

ABSTRACT

We describe possibilities of creation of virtual tours for virtual cities using viewpoint entropy method. If we want to obtain automatic virtual tour over the 3D scene, we can use low level off-line method (without any semantic information) called viewpoint entropy based on Shannon entropy for obtain the best views around objects and connect them with respecting of context of 3D scene. In this work we present method of selecting the best views around the objects in special cases, when the virtual scene is virtual city.

Keywords
Viewpoint entropy, virtual environment, virtual tour.

1. Introduction

In the field of computer graphics, one of the most important parts is creation of virtual environments. In the last years there is a big interest in creation of virtual cities. Man can visit a “real” city using various services (e.g. Google Earth, Microsoft Virtual Earth …). There is a question: How to create an automatic virtual tour with respecting virtual space of environment and best viewpoints for scene parts. In our method we combine viewpoint entropy for obtain the best views and context method for combine the fist *n* views into virtual tour through the scene. In our work we try to find method based on quality of view from geometrical data without any semantic information about parts of scene.

2. Quality of view measurement

We can explore 3D objects from different views with different quality of this views. This quality is very subjective so that’s the reason why is hard to quantify this quality [PPB+05]. Until now, there is no definition of good view in computer graphics, so there is no common quality criterion for “best views”. Intuitively we can put together quality of view with quantity of information from concrete view [VFSH01].

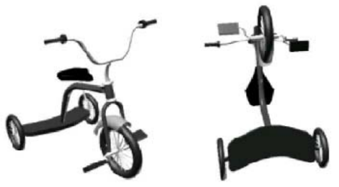


Figure 1: Two different viewpoints. Left is canonical. [BET94]

In [SPT06b] there is classification of different methods:

a) **Low level methods**: quantitative parameters of visible object parts
b) **Medium level methods**: geometric information about object
c) **High level methods**: semantic information about object

2.1 Low level methods

2.1.1 Viewpoint complexity

Plemenos a Benayada [PB96] bring in method based on number of visible surfaces and amount of visible surface. In [PS06] was presented simple extension of this method based on position of light sources in the scene.

2.1.2 Viewpoint entropy

Viewpoint entropy [VFSH01] is low level method based on Shannon entropy. Viewpoint entropy is ratio of relative sizes of visible triangles and surface of bounding sphere.

$$I(S,p) = -\sum_{i=0}^{N_f} \frac{A_i}{A_t} \cdot \log_2 \frac{A_i}{A_t}$$

Where *S* is scene, *p* is viewpoint, N_f is number of polygons in the scene and A_i/A_t is visibility of surface regarding to viewpoint *p*. In our method we use as bounding object hemisphere because in the virtual cities we are limited by ground. In the scenes where is no visible background is $A_0 = 0$, so A_i/A_t is increasing if surface *i* is visible from better angle and smaller distance.

2.1.3 Perspective frustum entropy

In many cases there is enough for our purposes to measure quality of view in one direction, so we don't need to project whole scene into the unit sphere.

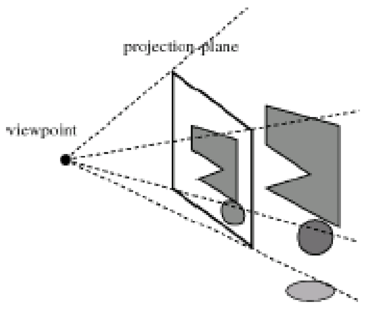


Figure 2: Perspective frustum entropy [VS02].

In this case we can use Perspective frustum entropy [VS02].

2.1.4 Relative entropy

In [SPFG05] was presented method based on Kullback-Leibler distance. The quality of view is highest when the amount of Kullback-Leiber distance is the lowest. The amount of KL distance is increasing in case that dimension of visible surfaces is the most different from real dimensions. In opposite to Viewpoint entropy we don't need to know number of visible surfaces and dimensions of background.

2.1.5 Viewpoint potential

This method was presented in [NTJ06]. It is useful method if we want to compute quality of view based on different properties such as viewpoint entropy, chrominance, luminance, weight of object, set of views and composition change

2.1.6 Visibility ratio

Man can calculate Visibility ratio as ratio between visible surface and whole surface of object. The goal is to show as much surface as possible.

2.2 Medium level methods

In low level methods there are important only two parameters: number of visible surfaces and quantitative parameters of surfaces. In Medium level methods we want to know also the curvature of visible surfaces (Curvature entropy [PPB+05], Mesh saliency [LVJ05]), silhouette (Silhouette length, Silhouette entropy and Topological complexity [PPB+05]).

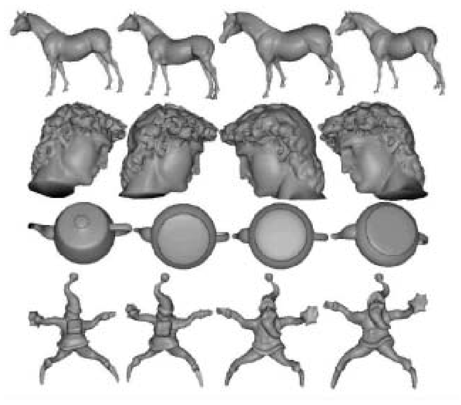


Figure 3: Curvature entropy [PPB+05]

2.3 High level methods

High level methods are based not only on the geometric or radiometric properties of objects but they need to know semantic of the object. For example in head model we can use segmentation for definition of which parts of the object represent nose, mouth, eyes, … Method based on this properties is Surface entropy of semantic parts. Similar method was presented in [SP06]. The main idea is to define relevation of each object or object part.

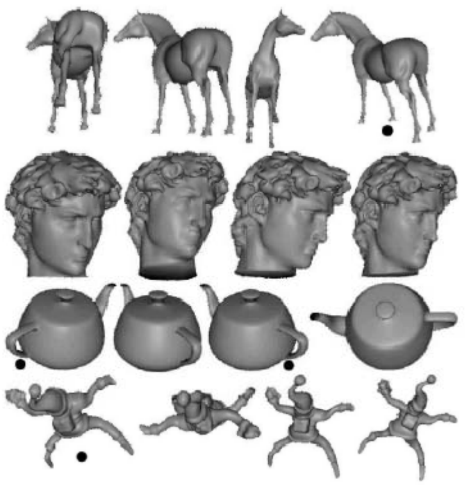


Figure 4: Surface entropy of semantic parts [PPB+05]

3. Exploration of virtual worlds

If we have large dataset and we want to automatically find path through scene, we can use some algorithms for exploration of 3D scenes. We can navigate through the scene in different ways. In many cases of creation of Virtual cities there is possibility to use panoramas as photorealistic demonstration of real world, but problem is navigation, because we are restricted only to jump between viewpoints. This problem can be solved by use real 3D model.

That's the reason for automatic method of exploration of 3D scenes. In [Ple03] these methods are divided into following groups:

a) On-line methods – user visit virtual world for the first time and camera path is locating incrementally. These methods need to compute camera path in real time.
b) Off-line methods – camera path is pre computed before visit of user and there is enough time to plan interesting views in the scene.

In our work we use Off-line method so we describe some important of them.

3.1 Global exploration

In global exploration techniques there are virtual camera still out of scene – at the virtual sphere around the scene. In [JTP06] is presented off-line global exploration method. In this method is scene inside the virtual sphere and is sampled into viewpoints. For each viewpoint compute quality of scene as a combination of visible surfaces and visible objects.

There are also other incremental on-line method presented in [BDP99] and [Vaz03].

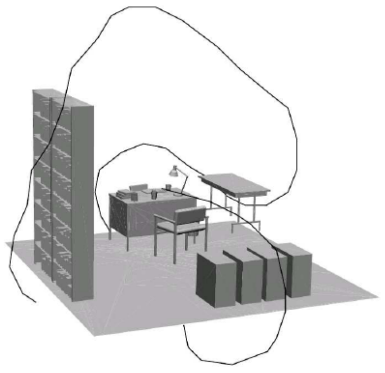


Figure 5: Camera path around virtual office [SPT06a]

3.2 Local exploration

In case of local exploration, camera can be a part of scene and can be too close to objects. Example of this problem is virtual museum. Basic rules for local exploration can found in [Sok06]:

a) Solution of collisions
b) Camera should visit the most important parts of scene
c) Quality of view should be the highest

In [VS03] is present method where camera is in constant height above the terrain and has only some degrees of freedom for movement. We use viewpoint entropy but only for surface which were not visible before and guided tour (virtual tour) is stopped after exploration of 80-90% of scene.

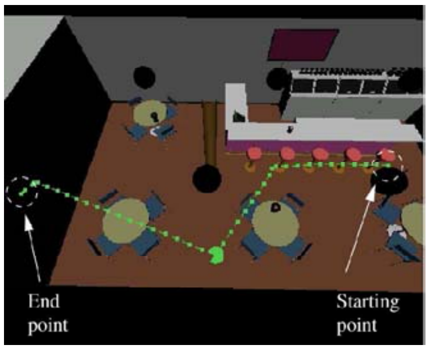


Figure 6: Berkeley soda hall local exploration [VS03]

4. Algorithm of context based method

Our algorithm is useful for large datasets of virtual cities. Today, there is many products for visualization of virtual cities, e.g. Google Earth, Virtual Earth and many other local cities e.g. Virtual Bratislava. In our method we try to present semiautomatic three step solution of creating virtual tours. We were inspired by movie techniques and real guided tours for real tourists.

In opposite to works [VS03] and [GAG04] we consider dividing of scene into objects as a kind of information. So we compare quality of view on one object with another view on the same object.

Our algorithm consist of these three steps:

a) Important scene parts selection
b) Path around each selected object
c) Final guided tour

In each of these steps we need to solve some problems. Scene is set of 3D objects with baseline in the same height (due to terrain).

4.1 Important scene part selection

This is manual part of whole process because we can't select the importance of objects from geometrical information. We can find some heuristic function as a solution of this problem but this is not good solution. This solution is based on choosing the biggest bounding box, … In this part we try to have semantic from geometry, but it is bad solvable problem.

4.2 Path around each selected object

In this part we compute viewpoint entropy for each object inside upper hemisphere around the object. Viewpoint entropy is handled for each camera position. We use perspective frustum entropy and we can select first *n* camera positions with the highest entropy. We need to have whole object always visible, so the perimeter of hemisphere is computed as

$$Dist(object, camera) = \frac{object.bbox}{2.\tan(FOV/2)}$$

Where *object.bbox* is diagonal of object bounding box and *FOV* is field of view for camera. If camera position at the sampled hemisphere lie inside bounding box of another object, we delete this position and we compute new one outside of the bounding box. In this case we don't need to have whole object visible. So we have steps in this part of algorithm:

a) Compute distance of object and camera (whole object have to be visible)
b) Sample the upper hemisphere (each sample represent camera position)
c) Compute quality of view for each camera position (using viewpoint entropy)
d) Construct path through viewpoints with the highest viewpoint entropy

4.3 Final guided tour

Last automatic step is join of different paths around objects into one path through scene. Properties of final path have to be no-collision with objects, C1 continuous and the shortest. Our method is for two objects. We can apply this for another pairs of objects in order. The most important role in this path planning has Voronoi diagram. If generated points are objects, then edges of Voronoi diagram are free path for camera. We consider three different approaches:

a) Additive weighted Voronoi diagram
b) Voronoi diagram for set of rectangles
c) Voronoi diagram of bounding boxes as approximation of b)

We choose for option c). So we have in this part these steps:

a) Create Voronoi diagram
b) Calculate guided tour for each selected object and test if camera position is not inside another object in scene.
c) Specify order of objects
d) Connect guided tours using Catmull-Rom spline curve.

In third step if we want to have the shortest path between objects we need to find Hamilton path what is NP complete problem so we use only heuristic for this problem.

If we want to connect path around selected object and edge of Voronoi diagram we try to find the shortest line with respecting camera position.

In our method there is possibility of adding some information to object (object weight) and for surfaces (surface weight) and use this information from e.g. reconstruction process to control the path of virtual tour through scene.

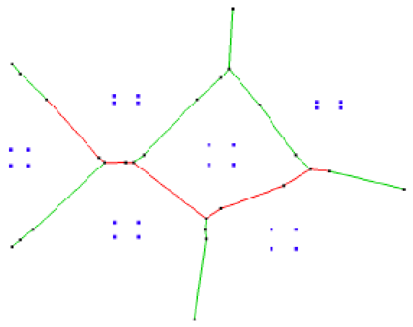


Figure 7: Shortest line between two vertices of Voronoi diagram

We use in our work only 2D Voronoi diagram, due to scene character. Scene is 3D buildings in city blocks. In fact, streets are edges of Voronoi diagram and blocks of buildings are generators of Voronoi diagram. For this reason we don't need to handle 3D Voronoi diagram, but if we will have more complex scene, we can use it for better results of camera path.

5. Results

We tested our algorithm with following computer specification: Arch Linux 64 bit, procesor AMD 4200+, RAM 2GB, graphic card NVIDIA 7300GT. We compute in window 512x512 pixels. We test speed of algorithm for 800 camera positions. Results are in following table:

Model	# of surfaces	time
Cube	12	8,44 s
Golfclub	515	8,55 s
Cow	5804	8, 95 s
Sphere	20480	10,30 s
Bunny	69666	14,08 s
Gipshand	273060	25,68 s
Dragon	871414	87,94 s
Buddha	1087716	106,77 s

Table 1: Time of compute the algoritm

As we can see in table time complexity of algorithm is linearly dependent on number of surfaces.

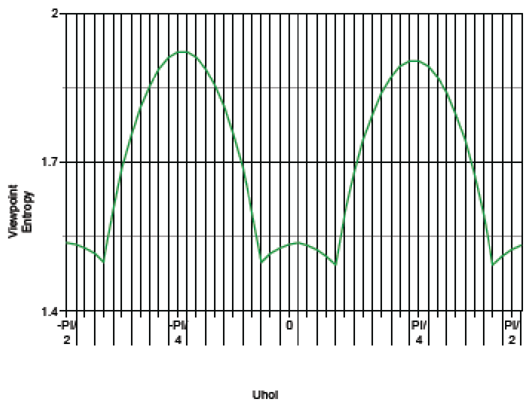


Figure 8: Graph of computation of viewpoint entropy for tested models

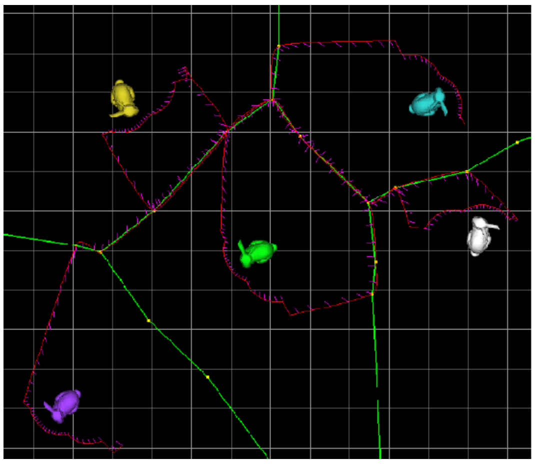


Figure 9: Final virtual tour through scene consist of bunny models (for complexity). Red – final path, green – Voronoi diagram

6. Future work

In future we plan to try different methods for quality of view based on pure geometrical information and also pixel based only in image space. Another step is to find methods for automatic selection of important scene parts (semantic from shape). Our work was specialized for virtual cities without another 3D models of architecture. We plan in the future use these small 3D models as a parts of generators of Voronoi diagram for improving camera path in the scene.

7. Acknowledgements

This paper was supported by VEGA grant

No. 1/0763/09

8. References

[BDP99] P. Barral, G. Dorme, and D. Plemenos. Visual understanding of ascene by automatic movement of a camera. GraphiCon, Moscow (Russia), 1999.

[BET94] Heinrich H. Buelthoff, Shimon Y. Edelman, and Michael J. Tarr. How are three-deminsional objects represented in the brain? Technical report, Cambridge, MA, USA, 1994.

[GAG04] Carlos Andújar Gran, Pere Pau Vázquez Alcocer, and Marta Fairén González. Way-finder: Guided tours through complex walkthrough models. Comput. Graph. Forum, 23(3):499–508,

[JTP06] Benoit Jaubert, Karim Tamine, and Dimitri Plemenos. Techniques for off-line scene exploration using a virtual camera. In International Conference 3IA'06, Limoges (France), May 2006.

[LVJ05] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. ACM Trans. Graph., 24(3):659–666, 2005.

[NTJ06] Machiko Nakagawa, Masami Takata, and Kazuki Joe. Automatic viewpoint selection for a visualization i/f in a pse. In E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, page 100, Washington, DC, USA, 2006. IEEE Computer Society.

[PB96] Dimitri Plemenos and Madjid Benayada. Intelligent display in scene modeling. New techniques to automatically compute good views. In GraphiCon, 1996.

[Ple03] Dimitri Plemenos. Exploring virtual worlds: Current techniques and future issues. In International Conference GraphiCon'2003, Moscow (Russia), September 2003.

[PPB+05] Oleg Polonsky, Giuseppe Patanè, Silvia Biasotti, Craig Gotsman, and Michela Spagnuolo. What's in an image? The Visual Computer, 21(8-10):840–847, 2005.

[PS06] Dimitri Plemenos and Dmitry Sokolov. Intelligent scene display and exploration. In International Conference GraphiCon'2006, Novosibirsk (Russia), July 2006.

[Sok06] Dmitry Sokolov. Exploration différéé de mondes virtuels sur Internet. PhD thesis, Université de Limoges, France, December 2006.

[SP06] Dmitry Sokolov and Dimitri Plemenos. High level methods for scene exploration. Journal of Virtual Reality and Broadcasting, 3(12), August 2006. urn:nbn:de:0009-6-11144, ISSN 1860-2037.

[SPFG05] Mateu Sbert, Dimitri Plemenos, Miquel Feixas, and Francisco González. Viewpoint quality: Measures and applications. In Eurographics Symposium on Computational Aesthetics, pages 185–192, June 2005.

[SPT06a] Dmitry Sokolov, Dimitri Plemenos, and Karim Tamine. Methods and data structures for virtual world exploration. The Visual Computer, 22(7):506–516, 2006.

[SPT06b] Dmitry Sokolov, Dimitri Plemenos, and Karim Tamine. Viewpoint quality and global scene exploration strategies. In Braz et al. [BJDM06], pages 184–191.

[VFSH01] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Viewpoint selection using viewpoint entropy. In VMV ’01: Proceedings of the Vision Modeling and Visualization Conference 2001, pages 273–280. Aka GmbH, 2001.

[VS02] Pere-Pau Vázquez and Mateu Sbert. Automatic keyframe selection for high-quality image-based walkthrough animation using viewpoint entropy. In WSCG, pages 461–468, 2002.

[VS03] Pere Pau Vázquez and Mateu Sbert. Automatic indoor scene exploration. In Proceedings of 6th International Conference on Computer Graphics and Artificial Intelligence 3IA’2003, pages 13–24, Limoges (France), May 2003.

[Vaz03] Pere-Pau Vázquez. On the Selection of Good Views and its Application to Computer Graphics. PhD thesis, Dept. LSI, Technical University of Catalonia, Barcelona, May 2003

Histogram Smoothing for Bilateral Filter

Michal Seeman
Faculty of Information Technology
Brno University of Technology
Bozetechova 2
CZ 612 66, Brno, Czech Rep.
seeman@fit.vutbr.cz

Pavel Zemcik
Faculty of Information Technology
Brno University of Technology
Bozetechova 2
CZ 612 66, Brno, Czech Rep.
zemcik@fit.vutbr.cz

Keywords
bilateral filtering, histogram, signal filtering, resampling

1. INTRODUCTION

Bilateral filtering was described and named by Tomasi and Manduchi [Tom00a]. The bilateral filter is a non-linear image filter that converts image into another image so that each pixel of the result is calculated as a nonlinear weighted average of the neighborhood of the corresponding pixel from the source image, where the weight drops both with spatial and intensity distance from the pixel from the source image pixel. In most cases, Gaussian is used as the weight function for both spatial and intensity domains.

Bilateral filtering is used in several graphics and image processing algorithms. One of the most important applications today is HDR processing mechanisms, specifically conversion of HDR images into 8-bit RGB images to allow their exploitation through standard display technologies..

Such HDR conversion uses bilateral filtering that can be accelerated through calculations of histograms of intensity values of local neighborhoods of pixels that are processed instead of individual pixels. So the histograms of intensity are gathered from the neighborhoods, convolved with some suitable smoothing function, such as Gaussian, and then used for calculation of pixel values of the output.

Several attempts have been made to accelerate bilateral filter computation [Dur00a], [Wei00a], [Por00a]. In our approach, we attempted using features of pixel local neighborhood. The essential part of the method is computing nonlinear filter approximation by using the local histogram convolution with the original filter's intensity domain Gaussian.

2. HISTOGRAM PROCESSING

Let *H(p)* be histogram of a neighborhood ε of a pixel *p*. Although we assume any bit depth of the source image, possibly even continuous (represented as floating point) values, the histogram can only be gathered into discrete bins which fact can potentially cause error. In our approach, the pixel contribution is subdivided into two nearest bins through linear interpolation. The error is discussed below.

Histogram Convolution

In the above mentioned application, the histogram *H(p)* should be convolved with Gaussian.

(1) $G_\sigma(x) = e^{-\frac{x^2}{2\cdot\sigma^2}}$

The traditional approach is to implement the convolution through direct numerical convolution with a convolution kernel corresponding to the Gaussian. However, in applications, where computational time is critical, such implementation is not suitable and can be improved. In the proposed approach the convolution with a Gaussian is approximated by the Exponential Moving Average (EMA).

(2) $EMA_\alpha(n,f) = \sum_{m=-\infty}^{n}(1-\alpha)^{n-m}\cdot f(m)$

EMA [Law00a] is a convolution of the function *f*(*n*) and an exponential curve $(1-\alpha)^{-m}$. The EMA convolution can be computed in linear time as shown below.

(3) $EMA_\alpha(n,f) = f(n) + (1-\alpha)\cdot EMA_\alpha(n-1)$

Note, that this definition of EMA is convolution with an exponential curve normalized by the first element, not by the sum of the elements.

The histograms have finite number of bins. As the proposed convolution core is defined from $-\infty$, it may seem that it would be required to expand the *f* definition and copy the first value *f*(0) to all undefined positions till $-\infty$ so that the initiation of the algorithm is done properly.

(4) $f(-1) = f(-2) = \cdots = f(-\infty) = f(0)$

Such expansion is, in fact, not needed as the infinite sum can be evaluated analytically and the EMA(0) can be easily precalculated.

(5) $EMA(0) = \frac{1}{\alpha}f(0)$

The complexity class of EMA is O(n) (where n is the number of histogram bins) comparing to the traditional convolution with a kernel whose complexity class is O(m×n) (where n is the number of histogram bins and m is the size of the convolution kernel). More importantly, the real computational time is much shorter than with the traditional approach.. This technique enables for computing convolution with a single exponential curve by a single and very simple loop through all items of $f(n)$. In fact, operation similar to convolution can be performed also in the reverse direction to compute convolution with an exponential curve mirrored along the y axis, such as: $(1-\alpha)^{+m}$. If the result of the mirrored function is moved one item forward and added to the non-mirrored result of EMA, the result is very close to the convolution with a symmetrical (mirror reflected) exponential curve.

(6) $$EMA2_{\alpha}(n,f) = \sum_{m=-\infty}^{n}(1-\alpha)^{n-m}\cdot f(m) + \sum_{m=0}^{\infty}(1-\alpha)^{n+m}\cdot f(m+1)$$

A single symmetrical EMA described above can be used to roughly approximate the convolution with Gaussian (as illustrated in Figure 1).

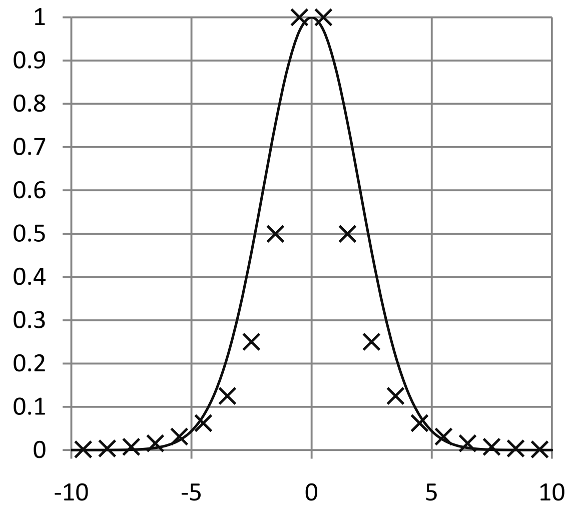


Figure 1. Gaussian σ=2 (continuous) and EMA2 α=0.5 (discrete)

Several instances of EMA can be used simultaneously in order to achieve even better results. As an example, let us present a method that exploits combination of three EMA2 convolutions through superposition to precisely approximate convolution with a Gaussian using σ=10.0 (see Figure2).

(7) $$3EMA(n,f) = 3.9\cdot EMA2_{0.150}(n,f) - 3.9\cdot EMA2_{0.247}(n,f) + 1.0\cdot EMA2_{0.387}(n,f)$$

The coefficients have been obtained through simple numerical minimization of error. Although the method is not perfect and could be possibly still improved, it already achieves very good results.

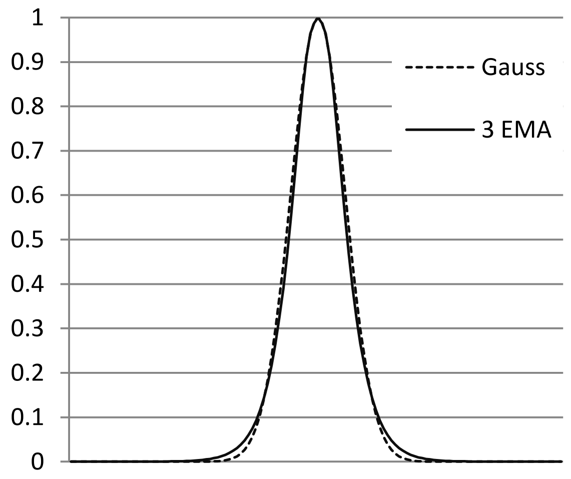


Figure 2. Gaussian σ=10 and it's approximation by 3EMA

The drawback of the presented method is that the convolution core center is between two samples (histogram bins), or more precisely, between the first two EMA samples in the mirror reflected core. It means that the convolution cannot be centered to the samples as it would be desired. This drawback, however, does not have too adverse practical implications.

As the histogram H is discrete, to obtain value of the convolved histogram for any value, an interpolation is be used. For our case, linear interpolation has been used. Possibly a better interpolation function that better reconstructs the discretized histogram values can be used, (e.g. Lanczos filter); however, the error caused by the simple interpolation is acceptable as shown further.

3. HISTOGRAM AND SAMPLING

As mentioned earlier, apart from the small error caused by imperfections in the Gaussian approximation, the histogram is also affected by the error caused by its discretization. To evaluate the error, let us describe the histogram processing using a processing pipeline.

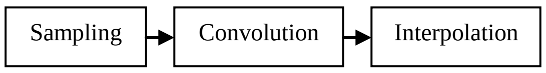


Figure 3. Signal pipeline

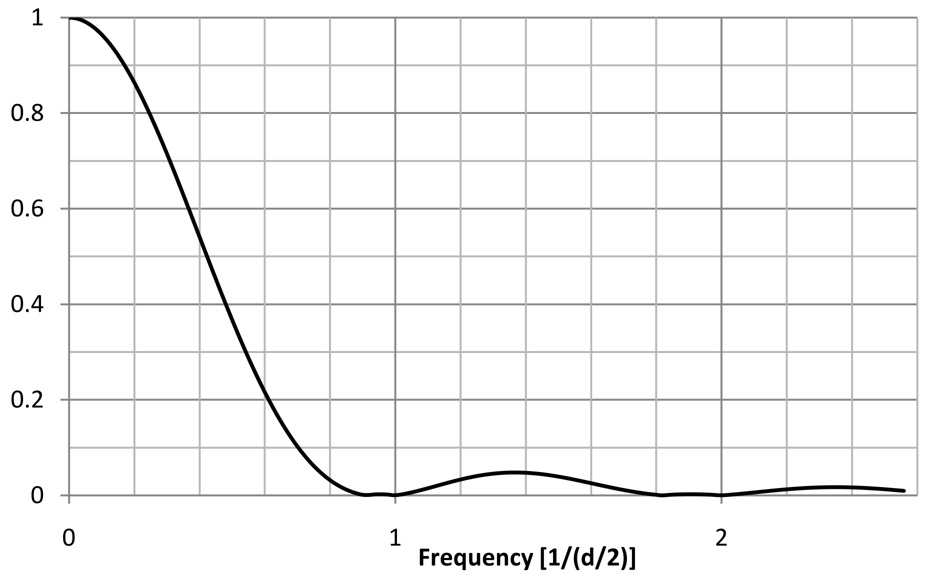


Figure 4 Triangle core spectra, frequency 1 wavelength is the bin spacing for linear interpolation – half of the triangle diameter(d)

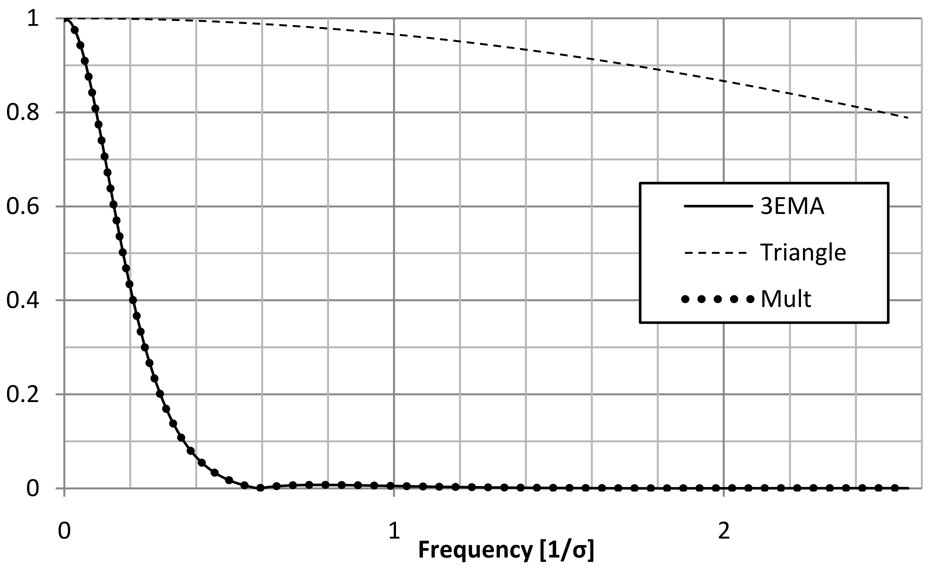


Figure 5. Compared spectra of 3EMA(σ=10), 2 bins wide triangle core and their product.The product is visually at the top of 3EMA

Both the initial sampling of the histogram with linear splitting to the bins and the final linear interpolation are approximately equal to convolution with a triangular core. Therefore, the result can be seen as a result of a sequence of convolutions. As this operation can be expressed in spectral domain using a point by point multiplication, the whole operation behavior can be illustrated by the spectra.

The Gaussian spectrum, as shown in Figure 5, is clearly a spectrum of a low pass filter. This is, in fact, the feature that enables discretization and subsampling of the histogram at all. As the result of convolution with Gaussian is a signal not containing too high frequencies, according to the sampling theorem [Nyq00a] it can be represented with relatively low number of samples whose density that correspond to the highest needed frequency in the signal – the sampling frequency must be over double of the maximum frequency contained in the signal. The minimum sampling frequency is, therefore, dependent on the parameters of the Gaussian. The question is, how does the initial distribution of the entries into the histogram among the samples and the final linear interpolation between the discrete convolved histogram values affect the result. Fortunately, as seen from the Figure 4 and Figure 5, the passband frequency of the Gaussian tends to be lower than that of the triangular core and as the result of the whole processing pipeline can be described in the spectral domain by point by point multiplication, it is clear that the result will not be affected significantly – the imperfections of the interpolation function in the spectra are not affecting the result as they fall into the part of the spectrum where they are multiplied with very close to zero values of the Gaussian itself.

4. PERFORMANCE

The proposed function 3EMA is an approximation of convolution with Gaussian with $\sigma=10$. It is important that the method doesn't approximate the convolution itself, but computes a precise convolution with an approximated core. There are two reasons:

1. In the mentioned HDR algorithm, two number sequences are to be convolved with Gaussian. The sequences are then merged. Possibly any Gaussian-like core could be used which would change the result quality according to the core accuracy. But it is essential to convolve both sequences with exactly same function.
2. To measure the method quality only the core quality has to be examined.

The core quality was measured as an average difference square (average of squares of differences between original Gaussian value and approximated core value). The average difference square for the 3EMA method measured on 128 item sequence (-64 to +63) is $3.26 \cdot 10^{-4}$.

Different combination of EMAs can be designed for specific maximum error required. The number of the histogram bins is then computed from the designed Gaussian σ size. The computation time is product of EMA curve count and signal sample count. Each EMA sample is computed using two multiplications and one addition; therefore the presented implementation of convolution is computed faster than the direct convolution implementation and also faster than the fast convolution techniques (that exploit the FFT and IFFT) [Nus00a].

5. CONCLUSIONS

This paper presented a novel approach high performance implementation of calculation of histogram convolved with Gaussian. Such calculations are essential e.g. for fast manipulation with high dynamic range images.

The new approach is based on the idea of subsampling of the histogram and implementation of the convolution core using exponential functions which lead into very fast implementation that outperforms the known solutions.

The new algorithm is of a O(n) class of complexity and generates very little error comparing to the traditional methods.

6. ACKNOWLEDGMENTS

This work was supported by the grant project of the Ministry of Educatuon, Youth and Sports of CR, (MSMT 2B06052) project “BioMarker”.

7. REFERENCES

[Tom00a] Tomasi, C., Manduchi, R. Bilateral Filtering for Gray and Color Images. in proceedings of the International Conference on Computer Vision, pages 839-846. IEEE, 1998

[Law00a] Lawrance, A., J., Lewis, P. A. W. An Exponential Moving-Average Sequence and Point Process (EMA1), Journal of Applied Probability, Vol. 14, No. 1 (Mar., 1977), pp. 98-113, Applied Probability Trust

[Dur00a] Durand, F., Dorsey, J.: Fast bilateral filtering for the display of high-dynamic-range images. ACM Trans. on Graphics 21 (2002) Proc. of SIGGRAPH conference

[Wei00a] Weiss, B.. Fast median and bilateral filtering. ACM Transactions on Graphics, 25(3), 519-526. Proceedings of the ACM SIGGRAPH conference, 2006.

[Por00a] Porikli, F., Constant Time O(1) Bilateral Filtering, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), June 2008 (IEEE Xplore, TR2008-030)

[Nyq00a] Nyquist, H., Certain topics in telegraph transmission theory, Trans. AIEE, vol. 47, p. 617-644, Apr. 1928

[Nus00a] Nussbaumer, H., J., Fast Fourier transform and convolution algorithms, Springer Series in Information Sciences, vol. 2, p. 287, 1982

Hypergraph-based software visualization

Peter Kapec

Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3
842 16, Bratislava, Slovakia

kapec@fiit.stuba.sk

ABSTRACT

In this paper an alternative approach to software visualization is presented based on hypergraph representation of software artifacts. Our approach builds upon known techniques that rely on graphs for visualization. Using hypergraph formalism offers significant advantages not only in the visualization process but also in data preparation, filtering and context retrieval. Our aim is to create a unified graphical environment capable to visualize relations between various levels of software related artifacts, from source code up to the project management tasks.

Keywords
Software visualization, software mining, hypergraph, zoomable user interface

1. INTRODUCTION

Understanding software systems is becoming more difficult due to the increasing complexity of currently developed systems. Because of intangibility of software, comprehending important aspects of software is difficult and often time-consuming. Software visualization tries to make software more "tangible" by providing visual representations. Using interactive visualizations we can achieve a better insight into software and probably reveal new and important information that would be difficult to obtain from textual source code.

Software is not only source code, but consist of many artifacts including data, algorithms, documentations, user interfaces etc. and all possible documents related to software development. These software artifacts occur in the whole development process. In current development environments it is difficult to track these artifacts and relations between them. Graphs and their visualizations are often used in the software visualization field, however approaches using hypergraphs are not very common. Our aim is to utilize hypergraphs to represent software artifacts, visualize these hypergraphs in an interactive way and to build an development environment based on hypergraphs.

2. SOFTWARE VISUALIZATION

Software visualization is an ongoing research direction. The usage of visualization of software can be tracked back to the beginnings of computer science. Early in the 1940-ties Goldstein and Neumann presented possible advantages of using flow-charts [Gol47]. The 1970-ties were mostly dedicated to Nassi-Schneiderman-Diagrams as an alternative to flow-charts. In the 1980-ties, when graphical workstations became available, new and more robust visualization systems could be developed. The 1990-ties were dominated by the attempt to utilize the third dimension. Although in past twenty-thirty years many software visualization systems were created, current software engineers still program in textual languages and use standard 2D GUI interfaces. Only few visualization systems managed to move from research projects to practice, which is mainly the problem of their evaluation in practice. However semantic software visualization is one of open challenges also mentioned in Koschke's study [Kos03].

Visual programming languages, a subfield of software visualization, often try to represent the source code through visual objects with some interconnections, which is similar to software visualization based on graphs. It has been shown that graph and hypergraph grammars can be used to define the syntax of visual languages [Bar99].

The contribution of software visualization in software comprehension was already shown [Lew02], but the usability of visual programming languages is very questionable [Gre92].

3. HYPERGRAPHS

Hypergraphs are generalized graphs in which an edge can connect more than two nodes. A formal definition of hypergraph follows:

Definition 1. *Let* $V=\{v_1,\ldots,v_n\}$ *be a finite set, whose members are called nodes. A hypergraph on V is pair* $H=(V,\varepsilon)$ *where* ε *is a family* $(E_i)_{i\in I}$ *of subsets of V. The members of* ε *are called edges.*

Graphs are often used to describe and visualize structured information where graph nodes represent information entities and edges represent relations between entity pairs. However in practice relations are often more complicated, often involving more than two entities and representing them trough graphs becomes difficult. Hypergraph edges allow to represent more complicated relations between several objects for those standard graphs would require several additional nodes and edges. Using hypergraphs it is possible to store knowledge about any domain – in our approach software is the domain. Knowledge representation deals with subjects and relations between them – in hypergraph terminology subjects can be represented using nodes and relations translate to hyperedges.

Usually hypergraphs are often represented using sparse incidence matrices with following definition:

Definition 2. *Let* $H=(V,\varepsilon)$ *be a hypergraph with* $m=|\varepsilon|$ | *edges and* $n=|V|$ *nodes. The edge-node incidence matrix of* H *is:* $M_H \in M_{m\times n}(\{0,1\})$ *and defined as:* $m_{i,j}=\begin{cases} \backslash & if v_j \in E_i \\ \cdot & else \end{cases}$

However we can modify the edge-node incidence matrix in definition 2 so that it does not contain just values 0 or 1. Replacing the 1 value with elements called incidences we get an alternative hypergraph definition that uses incidences [Aui02]:

Definition 3. *A hypergraph is a five-tuple* $H=(V,\lambda_V,E,\lambda_E,I)$ *where V, E, I are disjoint finite sets and we call V the vertex set of H, E the edge set of H, I the incidence set of H and* $\lambda_V : V \rightarrow P(I)$ *is a mapping that satisfies following conditions:*

$$\forall v \neq v' \quad \lambda_V(v) \cap \lambda_V(v') = 0 \quad \cup_{v\in V} \lambda_V(v) = I$$

and $\lambda_E : E \rightarrow P(I)$ *is mapping that satisfies the following conditions:*

$$\forall e \neq e' \quad \lambda_E(e) \cap \lambda_E(e') = 0 \quad \cup_{e\in E} \lambda_E(e) = I$$

Using this alternative hypergraph definition we can store additional information into incidences. Hyperedges usually represent undirected relations, however an incidence can specify a role the node plays in a relation. This hypergraph representation was used to formally define popular knowledge representation formats RDF and Topic Maps [Aui02] and similar concepts can be found in graph storage formats GLX [Hol00] and GraphML.

Using hypergraphs allows us to look at software as a knowledge repository with query functionality similar to databases. In our approach hypergraph nodes represent different software artifacts and hypergraph edges represent different relations between software artifacts. Nodes can represent source code artifact such as functions, classes, variables, types and other language constructs and documentations related to them, but can also represent subjects from development process such as developer names, specification documents, UML diagram entities, user interfaces, revisions, test data etc. As can be seen these artifacts are very different and even currently most complex development environments do not fully integrate them – usually they are stored in different file formats.

Mentioned artifacts are important, but for developers are relations between these artifacts more important. Looking at source code developers are interested in class inheritance, decomposition into modules, function call-graph, related documentations etc. However more important relations are not directly visible such as which developer implemented or modified what, how the specification influenced UML diagrams or e.g. how the function call-graph is related to class inheritance tree. Even more complicated relations can occur at program runtime or program debugging.

Currently heterogeneous programming environments are very common in development. Developers use more than one language for specific parts of the produced software. Using this approach programmers can leverage the execution efficiency of compiled languages, e.g. C/C++, while enjoying rapid development and flexibility of scripting languages. Often higher level interpreted languages access implementations in compiled languages and this crossing of language boundaries complicates software comprehension. As might be expected such approaches even more contribute to software complexity and make software comprehension difficult.

A hypergraph-based software artifact representation is capable to store also knowledge found in such heterogeneous environments by adding appropriate relations. The Figure 1 illustrates a hypergraph representation of a call-graph between three functions *f1, f2, f3*, where function *f1* calls functions *f2* and *f3*, and function *f3* calls *f1* and *f2*. Red spheres represent hyperedges representing *calls* relations, yellow spheres represent function nodes and green spheres represent incidences. Using incidences it is possible to distinguish the *caller/calle* roles in an undirected *calls* relation. The hypergraph representation hides the actual language implementation of mentioned functions, but captures their call relations.

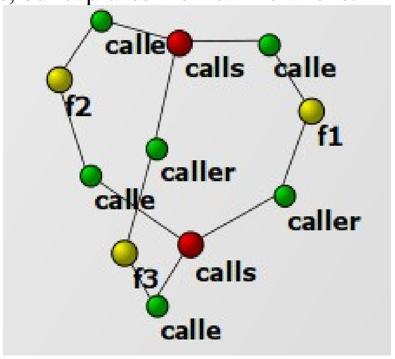


Figure 1. Call-graph between functions f1, f2, f3.

Similarly other software artifacts and relations of existing systems can be modeled.

4. HYPERGRAPH QUERIES

Displaying very large hypergraphs can be very confusing and not comprehensible, so some filtering is needed. We could use common graph traversal algorithms to filter graphs, but querying hypergraphs similarly to database queries offers more possibilities. We can define a hypergraph query language to query hypergraphs where query results are also hypergraphs. A query hypergraph consist of conditions for nodes, incidences and edges, and these conditions are searched in the queried hypergraph.

The query system is based on edge-queries, because edges represent important relations. However defining query hypergraphs visually would be time-consuming, it is possible to define queries in textual form. A notation in the form $E(I_1 : N_1, I_2 : N_2, \ldots)$ represents a one hyperedge query. In this textual representation E is the hyperedge name, I_1 represents the incidence connected to E and to node N_1, and similarly for other $I_i : N_i$ incidence-node pairs. The conditions can be regular expressions allowing complex search patterns.

To obtain the resulting hypergraph we follow a simple algorithm. For each edge in the query hypergraph we are searching for edges in the queried hypergraph. Edges are replicated to the result hypergraph only with the matching nodes and incidences that match with the query hypergraph. If any edge does not match the return hypergraph is empty. The Figure 4 shows a query hypergraph and results of this query.

5. VISUALIZATION

In our approach software artifacts and their relations represented through hypergraphs are visualized using spheres and lines in 2D space in which the user can navigate. Direct hypergraph visualization requires modifications to existing graph layout algorithms, however hypergraphs can be transformed into bipartite graphs using following definition:

Definition 4. *For a hypergraph* $H=(V,\varepsilon)$ *with an incidence matrix* M_H *the bipartite incidence graph* $B_H=(N_V \cup N_\varepsilon, E)$ *is defined as follows:*

$$E=\{\{m_i,n_j\}: m_i \in N_\varepsilon, n_j \in N_v, m_{i,j}=1\}$$
$$N_\varepsilon=\{m_i : E_i \in \varepsilon\}$$
$$N_v=\{n_j : v_j \in V\}$$

This transformation allows to utilize well known graph layout algorithms to visualize hypergraphs. In our approach we use a slightly modified force-based algorithm based on Fruchterman and Reingold graph layout algorithm [Fru91]. The modifications allow to pause the layout algorithm and to process only selected nodes independently. This allows the user to interactively interact with the visualization to place nodes of interest into preferred positions.

Hypergraph-based GUI

For programming tasks we need access to the underlying source code. For these tasks the interface can change individual nodes into floating billboards containing a 2D textual editor. The user can dynamically zoom into these billboards to perform programming tasks.

Figure 4 shows a hypergraph fragment containing the *installPackage* function and function's parameters. The function's node displays a text-editor with source code of the function that can be modified.

From software artifacts displayed in these billboards links to other nodes or billboards show different relations. Billboards are also affected by the force-directed algorithm, but can be forced into specified positions using meta-nodes and meta-edges that are user controlled and are not part of the visualized hypergraph. This allows to force position of e.g. documentation nodes to upper part of a window, billboards containing source code metrics e.g. into left window part and software artifact revisions into right window part. Other placements can be interactively created by user.

Example visualization of a system

To demonstrate the possibilities of hypergraph-based software visualization we analyzed a relatively small open-source project (80kB), but well documented and written in Lua language. We focused on obtaining not directly visible relations involving function call-graph, separation into modules, details about function parameters and return values and documentations related to these artifacts. Our initial analysis searched for only ten different node types and seven different hyperedge types. We extracted around 1200 nodes and 400 hyperedges, what are for such a small project relatively high numbers. Searching more node or hyperedge types would certainly dramatically increase the number of extracted artifacts and relations. For large projects we can expect even higher numbers of artifacts and relations.

A visualization of the whole extracted hypergraph is shown in Figure 3. This overview visualization is not very comprehensible, but we can see several highly connected hyperedges in figure middle part and a cluster. Focusing on this cluster reveals the core modules and functions responsible for main functionality. The highly connected hyperedges represent mostly *is-instance-of* relations, so naturally they connect nearly all nodes.

The figure 2 shows a filtered hypergraph containing only functions – result of the query i*s-instance-of(instance : * , type : function)*.

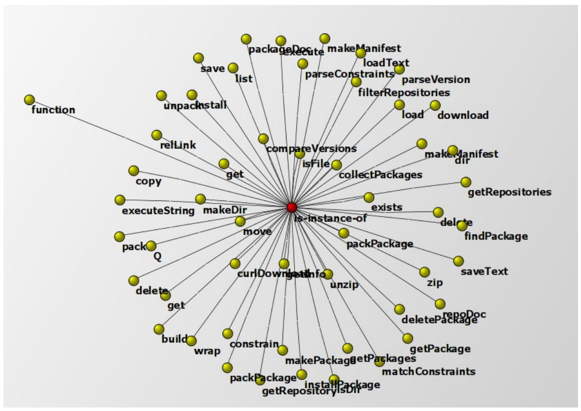


Figure 2. Filtered hypergraph.

The Figure 5 displays a query hypergraph (a) and results of this query (b). The query hypergraph receives all parameters of *installPackage* function and documentations related to this function and parameters. The query can be written in textual form as:

has-parameters(:installPackage, parameter:*) and*

is-documented(function : installPackage, short-description:) and*

*is-documented(function : installPackage, parameter : *, description : *)*

Using similar queries it is possible to obtain other visualizations of software.

6. CONCLUSIONS

The presented work is not necessary new and was inspired by several similar concepts. The Hypergraph Data Model [McB98] offers also a hypergraph-based storage of information, but uses two different hyperedge types and does not use a generic solution based on incidences. The Topic Maps knowledge representation format can be formally defined by hypergraphs, but it focuses mostly on semantic web. However hypergraph-based software visualization is not common. We presented visualizations of an existing software system and showed possibilities of hypergraph querying to obtain important information.

Current work is dedicated to the development of an hypergraph-based IDE that provides seamless integration of hypergraph visualization and conventional textual programming. Ideas for future work include hypergraph visualization in 3D with different hypergraph layout algorithms and visualization of programs at runtime. Other possible research directions include collaborative hypergraph editing in virtual environments.

7. REFERENCES

[Aui02] Auillans, et al., A formal model for topic maps. In ISWC'02: Proceedings of the First International Semantic Web Conference on The Semantic Web, Springer Verlag, pp. 69-83, 2002

[Bar99] Bardohl, R., Minas, M., Schurr, A., Taentzer, G. Application of graph transformation to visual languages, 1999.

[Gre92] Green, T.R.G., Petre, M. When Visual Programs are Harder to Read than Textual Programs. In: Human-Computer Interaction: Tasks and Organisation, Proceedings ECCE-6 (6th European Conference Cognitive Ergonomics), 1992.

[Fru91] Fruchterman, T. M. J., Reingold, E. M. Graph drawing by force-directed placement. Software – Practice & Experience, 21(11):1129–1164, 1991.

[Gol47] Goldstein, H. H., Neumann., J. von. Planning and Coding Problems of an Electronic Computing Instrument. In: Taub, A.H., von Neumann, J., Collected Works, pp. Conclusions 80-151, McMillan, New York, 1947.

[Hol00] Holt, R. C., Winter, A., Schurr, A. GXL: Towards a Standard Exchange Format Universitat Koblenz-Landau, Institut fur Informatik, Rheinau 1, D-56075 Koblenz, 2000

[Kos03] Koschke, R. Software Visualization in Software Maintenance, Reverse Engineering, and Reengineering: A Research Survey. In Journal on Software Maintenance and Evolution, John Wiley & Sons, Ltd., Vol. 15, No. 2, pp. 87-109, 2003.

[Lew02] Lewerentz, C., Simon, F. Metrics-based 3D Visualization of Large Object-Oriented Programs. Proceedings of the First International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT02), 2002.

[McB98] McBrien, P., Poulovassilis, A. A General Formal Framework for Schema Transformation. Data and Knowledge Engineering 28, pp. 47–71 1998

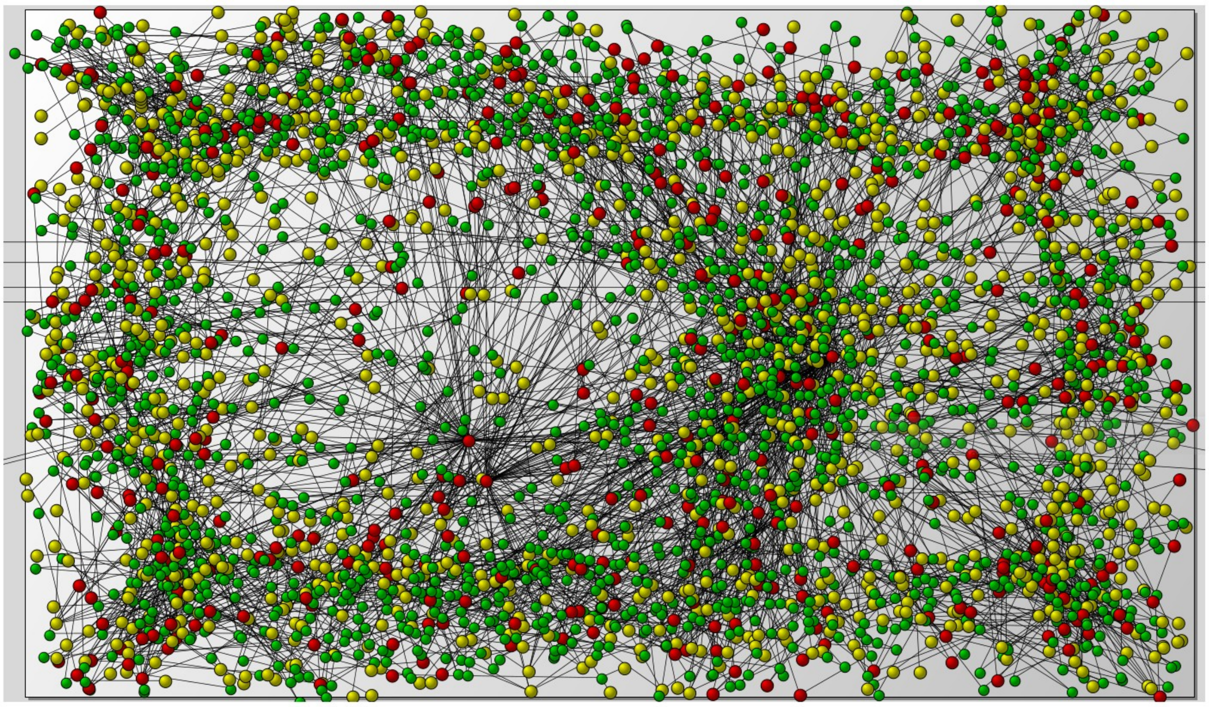
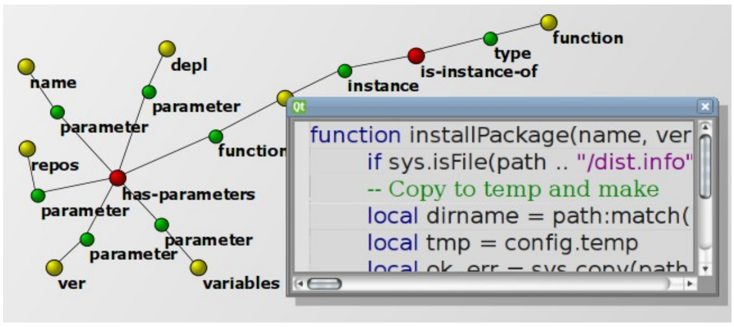


Figure 3. Visualization of whole extracted hypergraph.



```
function installPackage(name, ver
    if sys.isFile(path .. "/dist.info"
    -- Copy to temp and make
    local dirname = path:match(
    local tmp = config.temp
    local ok, err = sys.copy(path
```

Figure 4. (a) Text editor attached to a node.

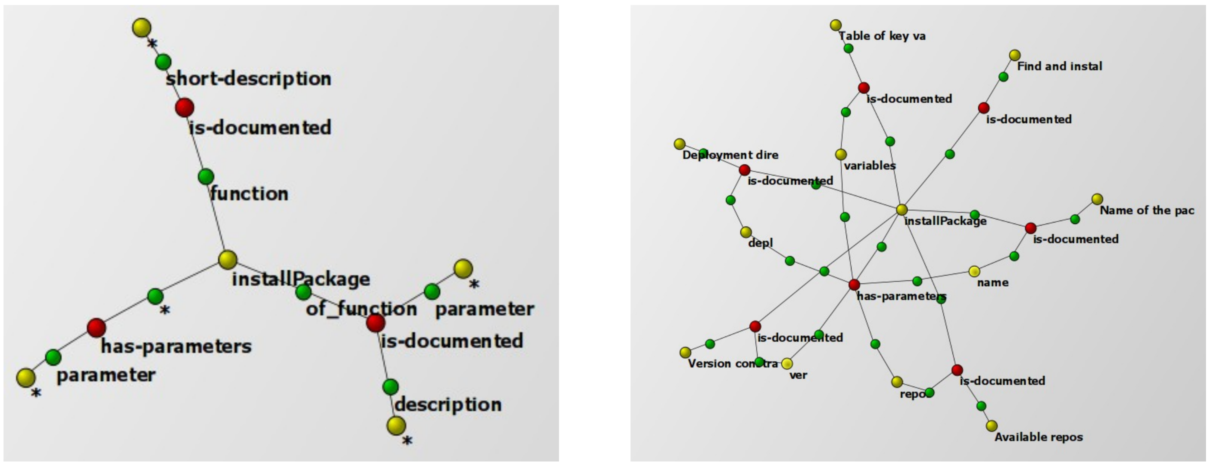


Figure 5. (a) Query hypergraph (b) results of query hypergraph.