Advances in Metric-neutral Visualization

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

ABSTRACT

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

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

1 INTRODUCTION

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

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

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

1.1 Comparison with previous work

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

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

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

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

2 NONEUCLIDEAN GEOMETRY

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

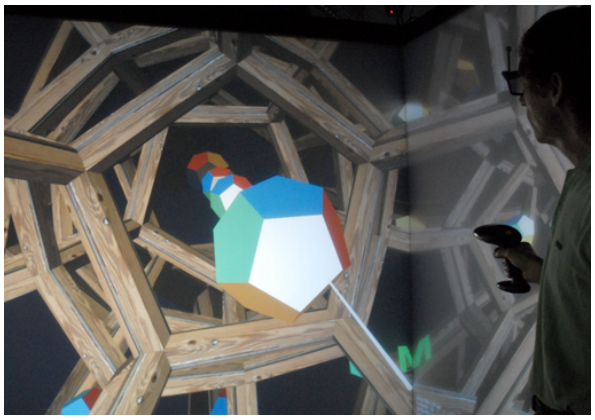

Figure 1: Immersive experience of elliptic space.

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

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

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

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

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

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

2.1 Isometry groups

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

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

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

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

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

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

2.2 Elliptic space and spherical space

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

3 NONEUCLIDEAN VISUALIZATION

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

3.1 The insider's view

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

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

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

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

3.2 Cameras are projective, not metric

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

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

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

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

3.3 Visualization issues with elliptic space

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

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

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

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

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

4 METRIC-NEUTRAL INFRASTRUCTURE

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

4.1 Infrastructure challenges

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

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

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

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

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

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

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

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

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

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

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

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

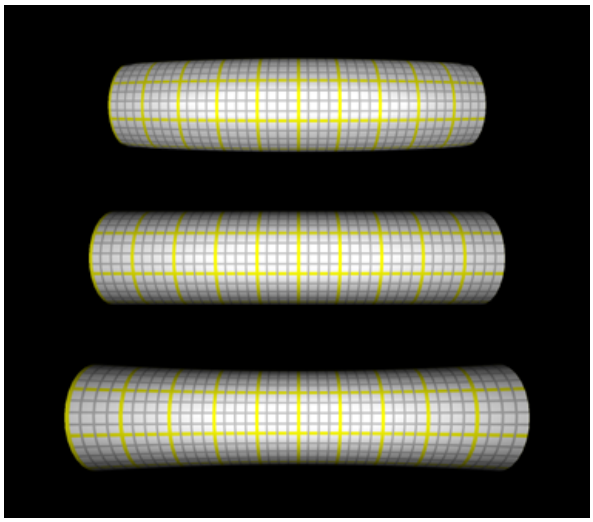

Figure 2: Hyperbolic, euclidean, and elliptic tubes around a horizontal line segment

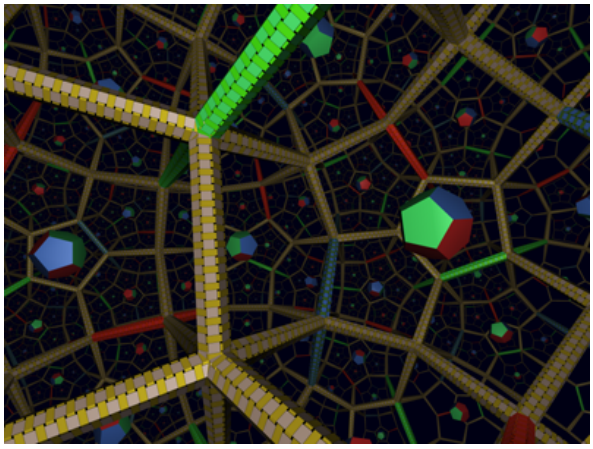

Figure 3: Real-time hyperbolic shading implemented with an OpenGL Shading Language vertex shader

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

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

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

and other effects involving distance and angle measurement.

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

4.2 Implementation details

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

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

4.3 Geometric algebra

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

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

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

5 METRIC-NEUTRAL INTERACTION

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

5.1 Metric-neutral Picking

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

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

5.2 Mouse-based tools

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

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

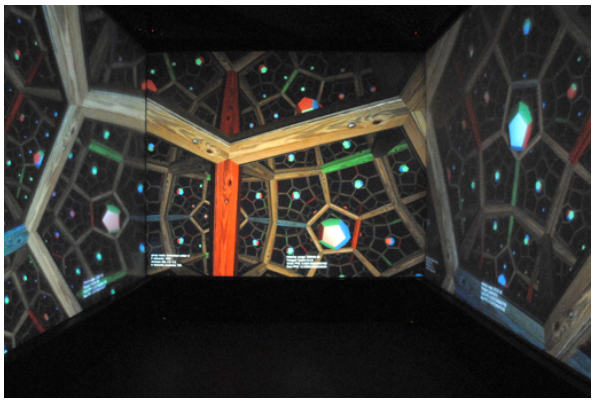

Figure 4: Immersive experience of hyperbolic space in a 3-walled CAVE.

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

6 METRIC-NEUTRAL IMMERSIVE ENVIRONMENTS

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

6.1 Wand-based tools

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

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

6.2 Metric-neutral tracking

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

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

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

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

6.3 Unit lengths and the scaling problem

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

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

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

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

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

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

6.4 Metric-neutral tracking

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

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

6.5 Discussion

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

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

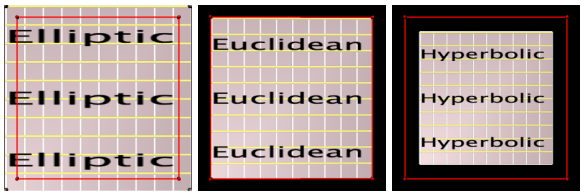

Figure 5: How the front wall appears from the middle of the back wall with a unit length of 1.73m, in the three metrics. Red outline is the physical wall.

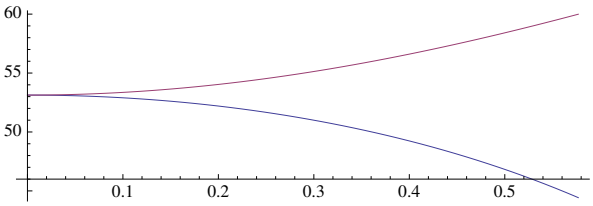

Figure 6: Graph of visual angle in degrees (vertical axis) vs. scaling factor (horizontal). The elliptic case is the red curve; the hyperbolic, blue.

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

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

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

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

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

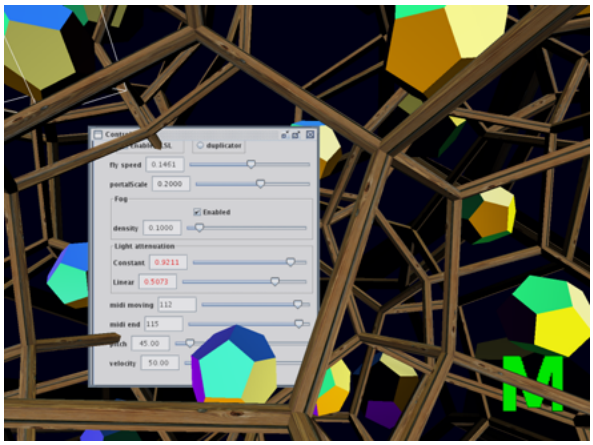

Figure 7: 2D Java GUI embedded as euclidean element in elliptic space.

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

6.6 Metric-neutral immersive 3D GUI

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

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

7 FURTHER WORK

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

8 CONCLUSION

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

A CAYLEY-KLEIN METRICS

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

A.1 From quadratic form to projective metric

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

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

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

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

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

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

A.2 Polarity on the metric quadric

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

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

A.3 Distance and Angle Formulae

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

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

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

The familiar euclidean distance between two points:

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

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

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

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

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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