EXPERIMENTAL GEOMETRY LAB AT THE UNIVERSITY OF MARYLAND

WILLIAM M. GOLDMAN

The Experimental Geometry Lab

Founded in 2000 by W. Goldman and R. Schwartz, the EGL has been a focus for interactive projects in geometry research at the undergraduate level. After Schwartz resigned from the Maryland faculty, the EGL has been run by Goldman alone, together with a (rotating) group of advanced graduate students and postdocs. The goal of the EGL was to develop a community of mathematicians at all levels involved in experimental investigations of geometric structures (broadly defined) in dimensions 2 and 3.

Mostly active in the summers, when students have freer schedules to pursue independent study, the EGL has trained about 5-10 students on the average each year. The software projects are available and are documented on the EGL web page `http://www.egl.umd.edu`, although the website is somewhat out-of-date.

Our vision is that these software packages would be easily accessible on the internet so that a computer user surfing the web might encounter EGL projects, and start accessing them. Hopefully the cool graphics and intriguing animations would pique the interests of this hypothetical user, who would be tempted to learn more about the subject (mathematics). The software would be full of demos, which can be easily tweaked to explore variations of these projects. Some users might be tempted to actually get into the code and write and expand the existing programs. This would provide an invitation to cutting-edge mathematical research topics, which serves as both a fascinating and satisfying way to learn mathematics.

Our philosophy so far in these programming projects has emphasized *commincation.* The programs tend to be very short, but very dense. Much content is packed into just a few lines of code. Therefore I have encouraged a student to liberally document and comment their code and provide many demos and examples. (At one point I suggest about 90% comments to ever 10% of actual code). Easily understandable,

Date: August 25, 2016.

modular code, is important both for expediting the debugging process, as well as making code which can extend to larger projects.

The EGL format which seemed to work best involved one or two advanced graduate students or postdocs, who ran an intensive course for the students, often beginning undergraduate math majors, to teach the students basic mathematics needed for the software projects. The software projects have so far all been in areas vaguely related to Goldman's research interests: geometric structures on manifolds and discrete groups. However, they involve material (such as geometric group theory, non-Euclidean geometry, advanced linear algebra) which, although basic, are not part of the standard beginning undergraduate curriculum.

Recently *Scientific American* reported on the EGL, its formation and its export to other insitutions by former members: see:

`http://www.scientificamerican.com/article.cfm`
`?id=deep-spaces-geometry-labs`

An example of mentoring

In 2007, Rachel Kirsch, an undergradate math major at Maryland, began a project in the EGL, jointly supervised by then-graduate student Ryan Hoban. Using the hyperbolic geometry packages and combinatorial group theory she drew illustrations of subtesellations of genus-two surface groups inside triangle tesselations (Figures 1 and 2). This involved creating algorithms inside a noncommutative group (the group generated by refletions in the sides of the triangle) rather than explicit numerical computations with matrices in hyperbolic geometry. Working at this high-level she learned abstract ideas from geometric group theory and how to apply to them to a specific concrete problem. Her illustrations were used in my paper *Higgs bundles and geometric structures on surfaces,* in *The Many Facets of Geometry: a Tribute to Nigel Hitchin,* O. García-Prada, J.P. Bourgignon, and S. Salamon (eds.), Oxford University Press (2010) 129 – 163, `math.DG.0805.1793`.

Rachel continued her study of mathematics in the doctoral program at the University of Nebraska, where she is studying combinatorics.

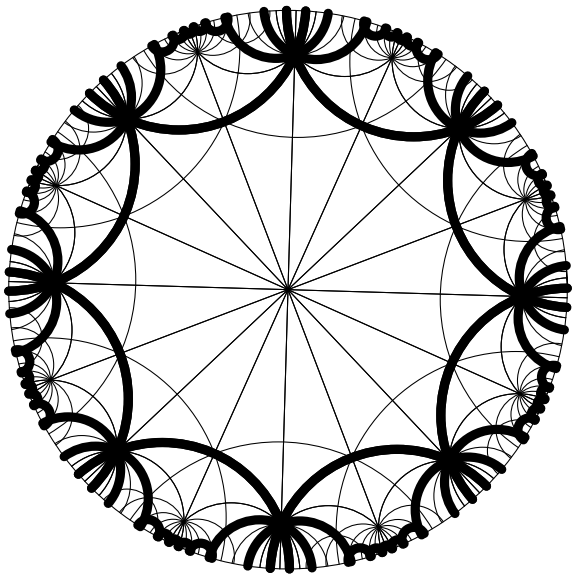


FIGURE 1. Subtesselations by 45-degee regular hexagons

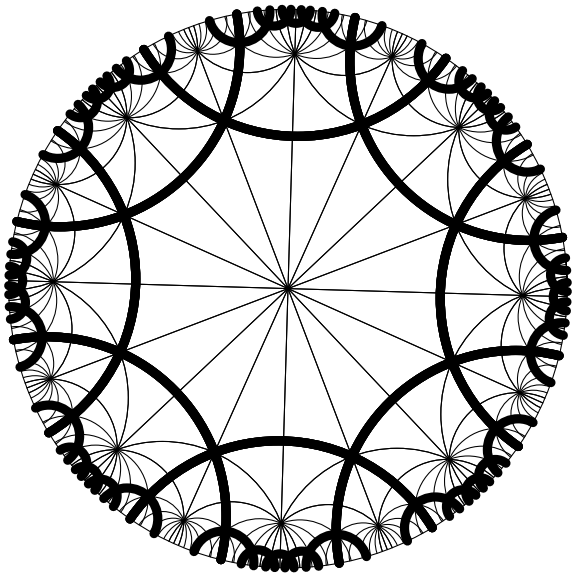


FIGURE 2. Subtesselations by 90-degee regular hexagons

Origins of the project

I had been developing software for visualizing geometric structures since 1984 when at MIT I programmed a Commodore 64 to draw convex real projective structures on surfaces. (Compare Figures 3, 4 and 5.

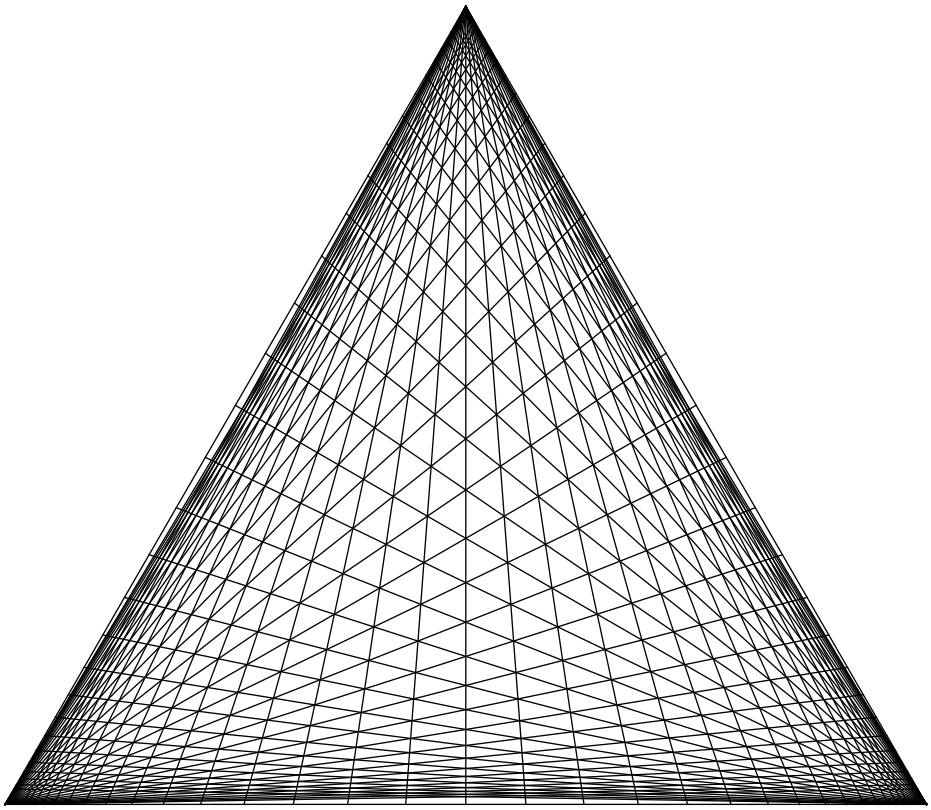


FIGURE 3. Projective deformation of a Euclidean triangle tesselation

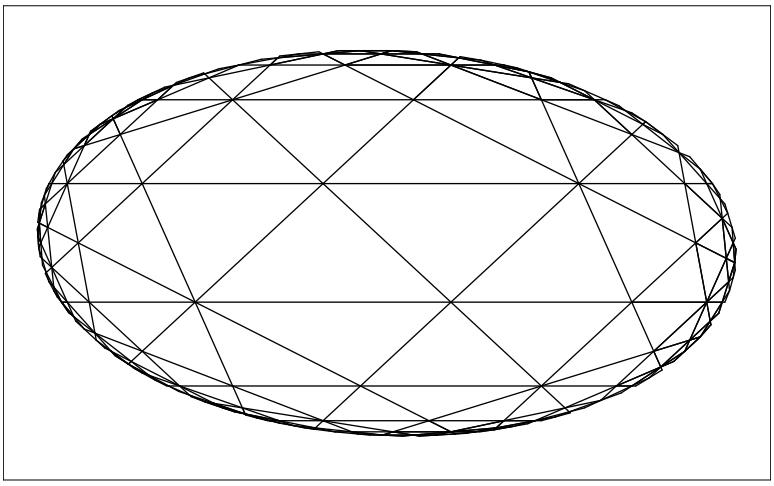


FIGURE 4. A hyperbolic triangle tesselation in its projective model

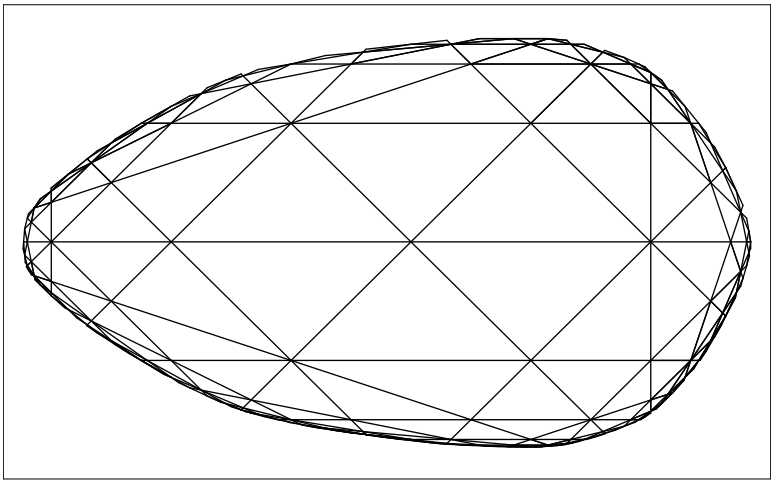


FIGURE 5. Projective deformation of a hyperbolic triangle tesselation

I continued this when I moved to Maryland in 1986 and produced computer-generated illustrations for my book "Complex Hyperbolic Geometry," Oxford Mathematical Monographs, Oxford University Press xvii + 316 pp. + 58 illus. (1999). Significant development occurred at the Geometry Supercomputer Project at the University of Minnesota. With then-graduate students Mark Phillips and Robert Miner, we developed a visualization program *Heisenberg* to draw pictures on the boundary of complex hyperbolic space. We produced two VHS movies, called *A tour of Heisenberg space,* and *Complex hyperbolic Kleinian groups* to illustrate the spherical CR-geometry on the Heisenberg group bounding complex hyperbolic space.

At an early stage I learned the quirkiness of these projects. We were developing the software tools simultaneously with developing the mathematics. In the complex hyperbolic geometry project, I was building on some software written by Thurston student Silvio Levy to draw limit sets of complex hyperbolic quasi-Fuchsian groups, articles by Mostow, books by Helgason and Rudin, and a paper written by É. Cartan. All of these references used different conventions so in order to blend the ideas together I had to choose one standard convention for the computations. Then I learned a great deal of mathematics by proving the theorems from our source using the techniques and viewpoints of the other. Much of the new mathematics found in *Complex Hyperbolic Geometry* was developed using this technique.

An interesting example of this style of working was the pair of totally geodesic foliations of bisectors in complex hyperbolic space. Mostow showed that bisectors could be understood as a pencil of complex hyperplanes ("slices") along an orthogonal geodesic ("the spine"). I developed formulas for their boundaries ("spinal spheres") in Heisenberg space and proceeded to draw them on a machine. I noticed similarities between these and the formulas I was using to draw $\mathbb{R}$-circles (boundaries of $\mathbb{R}$-planes, totally real totally geodesic surfaces in $\mathsf{H}^2_{\mathbb{C}}$). These similarities led to the realization that bisectors decomposed into $\mathbb{R}$-planes as well as complex hyperplanes. This elementary result became a cornerstone of the theory developed in my book *Complex Hyperbolic Geometry.* Some of the relationship between the expeerimental approach and the traditional approached is discussed in the introduction of that book.

Figures 6 and 7 illustrate the geometry of the boundary of complex hyperbolic space (the conformal geometry of the Heisenberg group).

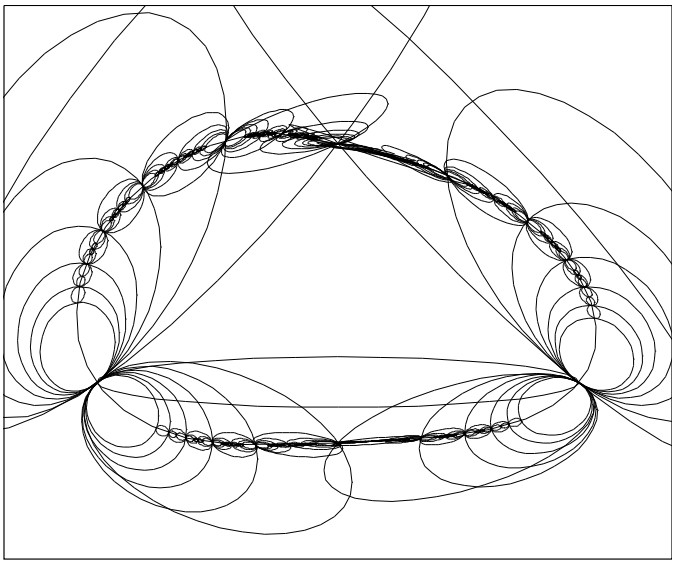


FIGURE 6. An ideal triangle group in Heisenberg geometry

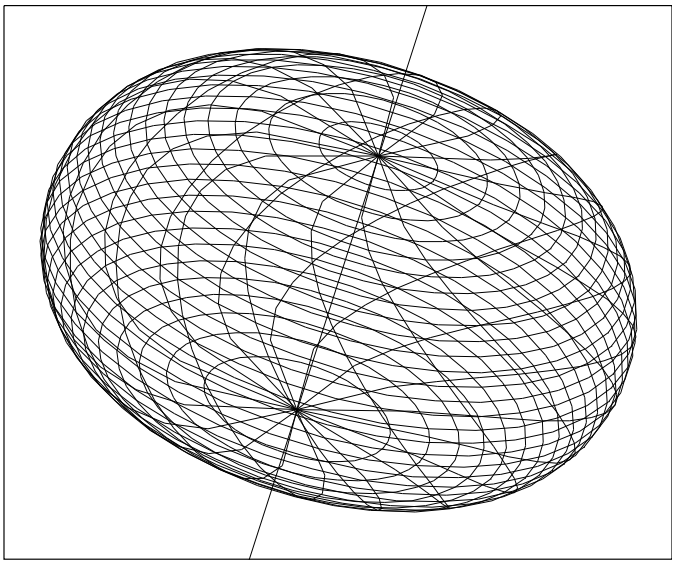


FIGURE 7. The boundary of a bisector in Heisenberg, foliated in two ways

More Geometry Labs

Anton Lukyanenko joined the EGL at the end of his freshman year at Maryland, and soon he began supervising projects and administering the lab. After completing his Bachelors degree at Maryland, he stayed on and completed his Masters degree in 2008. He then moved to the doctoral program at UIUC in 2008, where he and Jayadev Athreya established the *Illinois Geometry Lab* in 2010. This project was extremely successful with enormous participation from many different faculty members and students. After completing his doctorate in 2014 under Jeremy Tyson, he moved to the University of Michigan as a postdoc. Jayadev Athreya has since moved to the University of Washington, and now Jeremy Tyson is running the Illinois Lab. Now Lukyanenko and Athreya are in the process of starting labs in Ann Arbor and Seattle, respectively.

Sean Lawton completed his doctorate in 2006 at Maryland, and returned in 2009 as a postdoc. His affiliation with the EGL began earlier, while he was a graduate student, helping with numerous outreach activities. During that year he and then-graduate student Ryan Hoban ran an intensive summer REU-type project in the EGL. After that, he moved to UT-RGV and established a Geometry Lab (EAGL) there in 2009. In 2014 he moved to George Mason University, where he and his colleague Christopher Manon extablished the *Mason Geometry Lab* (MEGL).

David Dumas at the University of Illinois at Chicago has established a Geometry Lab there.

With the proliferation of these activities, the organizers banded together and formed *Geometry Labs United,* which held its first conference in September 2015. In addition to distinguished mathematical researchers whose work had an experimental component, there was a workshop on how to establish similar projects.

Recent activities at Maryland

The Experimental Geometry Lab has been useful in several projects in the recently established MAPS-REU program in the Maryland Mathematics Department. Also several graduate students have developed visualization tools, such as Mathematica Notebooks and 3d prints, to aid their research.

Recently Goldman has been developing a course (*MATH/AMSC 431: Geometry for Computer Applications*) which teaches mathematics to undergraduates interested in computer graphics, robotics and computer vision. Starting from a background in calculus and linear algebra,

the course the course develops fundamental mathematics necessary for computer graphics, robotics, computer vision and other applications of geometry:

- Projective geometry: the mathematics of perspective;
- Vectors, matrices and their geometric interpretations;
- Geometric transformations (rotations, reflections, translations, projections);
- Homogeneous coordinates, and data types for points, lines and planes;
- Conic sections;
- Complex numbers and quaternions;
- Topology.

Up to now, the course (which has run six times) has not involved many programming projects. A future direction for the EGL is to develop interactive tools (for example in *Geogebra*) to illustrate these concepts.

The advent of 3-dimensional printing has opened up a new direction in these projects. Current doctoral student Jean-Philippe Burelle developed models of fundamental domains for Margulis spacetimes (complete flat Lorentzian 3-manifolds with free fundamental groups), which are illsustrated in Figures 8,9.

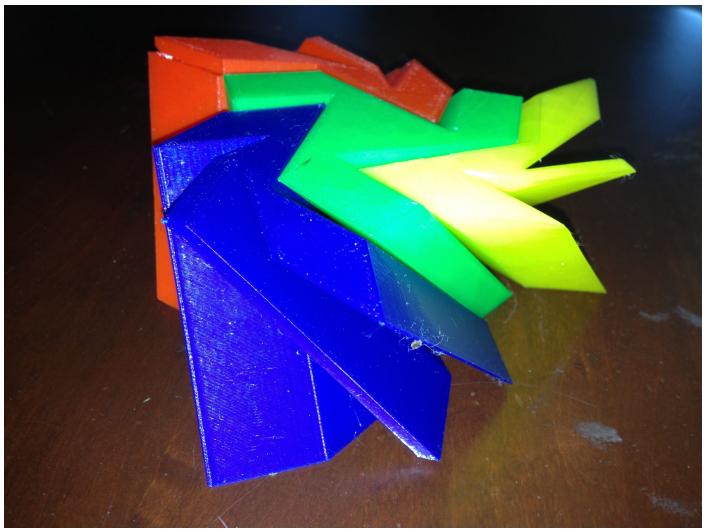


FIGURE 8. A crooked polyhedron

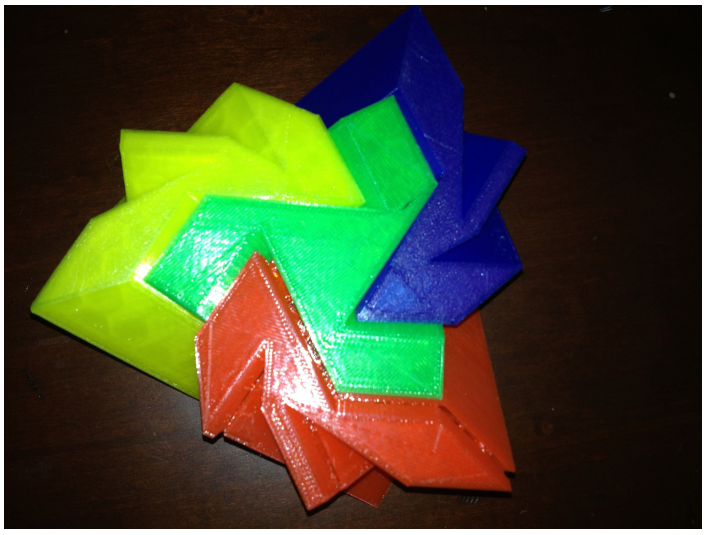


FIGURE 9. Head-on view of crooked polyhedron

Overview of Math 431, Fall 2015

by Caleb Ashley and Bill Goldman

This course develops the theory of data types for computer applications. Specifically we develop algebraic data types for computer graphics, computer vision and robotics. If we take points, lines, lengths, angles, areas, etc. as the (extremely basic) building blocks of graphics, then an over arching theme is to discover ways in which to represent these geometric objects in a computer. Simultaneously, we aim to develop ways in which to represent in a computer the manner in which these geometric objects transform.

—Imagine designing a video game—

Your avatar is flying a spaceship through a hazardous jungle populated by wild monsters, evil dinosaurs and poisonous plants, with dangerous objects zipping by. Throw in a few earthquakes, tsunamis, hurricanes and tornados, too, just for fun.

Of course enemies are chasing you, shooting rockets and subjecting your craft to waves of treacherous force fields. In addition to steering your vehicle, you need to be able to change your viewpoints and perspectives in order to fully gauge your direction and speed. Your instruments need to sense all the awful perils your adversary has aimed at you. Your survival, and the lives

> of millions of other people, depends on being able to manipulate — reliably, quickly, and in real time — huge amounts of graphical data by many types of geometric transformations: rotations, dilations, translations, reflections, and changes of perspective.

Goals

- Due to total size of the graphical data, the data types must be compact and efficiently designed.
- Due to the demands of interactive use, the computations must be as fast as possible.
- Due to the demands of ever-changing technology, the code must be easy to debug, maintain, and update. Thus the data types and the manipulation routines must be readable, succinct and comprehensible to other programmers.
- The data will ultimately be vectors and matrices, and the mathematical routines basically linear algebra. Matrix operations are cheap, efficient and easy to implement. The compelling advantage of linear transformations is that —by using coordinates on a vector space defined by a basis — the geometric information is encoded in a *finite set* of numbers. It's only how they are manipulated which varies by their context.

Here are some examples of how abstraction and mathematical elegance are both means and end in regards to computer applications.

Data types in plane geometry

First consider the familiar case is that of points in the plane. Points are described uniquely by an ordered pair of numbers. Vector operations enable us to compute geometric relations (such as distance) between points. Furthermore transformations of the plane are conveniently described by matrices. The calculations are cheap to implement on a computer, easy to understand.

Trying to do the same for lines in the plane is more difficult and more interesting. However, programming a video game may require you to transform lines and, eventually, more complicated graphical objects, in a similar way. The reality is that lines in the plane are not as easy to parametrize as points in the plane: the set of lines does not admit a coordinate system as easy as just the (x, y)-coordinates which uniquely describe arbitrary points.

Here are some ways we describe lines in the plane. As you can see, none of them are as convenient as parametrizing points in the plane.

(1) Given two distinct points p_1 and p_2 (represented as 2-vectors), the line $\overleftrightarrow{p_1p_2}$ joining them is described by equations

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

or, in parametric form,

$$x = x_1 + t(x_2 - x_1)$$
$$y = y_1 + t(y_2 - y_1)$$

This parametrization has the following drawbacks:

- The points p_1, p_2 are not unique and may not be so efficient to find; there are many pairs of points which determine a given line.
- Determining when two different pairs (p_1, p_2) determine the same line may be unnecesarily time-consuming.
- The initial data requires that $p_1 \neq p_2$. Checking this each time is necessary (and time-consuming).

(2) Given a slope $m \in \mathbb{R}$ and $b \in \mathbb{R}$, the line with slope m and y-intercept b has *slope-intercept form*

$$y = mx + b.$$

This efficiently parametrizes all *non-vertical* lines uniquely by the pair $(m, b) \in \mathbb{R}^2$. However, vertical lines have "infinite slope" ($m = \infty$) and don't fit nicely into this parametrization. However they are parametrized by the x-intercept $a \in \mathbb{R}$: the vertical line corresponding to $a \in \mathbb{R}$ is given by $x = a$.

(3) One can remove (or, more accurately, "hide") the difficulty with infinite slope by replacing the slope m by the *angle of inclination,* that is, the angle θ the line makes with the x-axis. These two parameters relate by:

$$m = \tan(\theta)$$

However, like all angles, θ is only defined up to multiples of π. One can restrict θ to lie in an interval, say, $0 \leq \theta < \pi$, but this line does not vary continuously as $\theta \nearrow \pi$.

(4) Lines can also be parametrized by their *closest-point-parameters* as follows. A line L contains a unique point p which is closest to the origin O. If $p \neq O$, then this point determines L uniquely. However, the case $p = O$ (that is, when $L \ni O$) has to be handled separately, like the vertical lines in the slope-intercept parametrization.

Topology is the villain

The problem cannot be solved easily, since it reflects a fundamental fact, involving the *topology* of the set of lines in the plane. Unlike the points in the plane, which form a tractable algebraic object (a *vector space*), the lines in the plane form a space which is inherently more complicated. "Topology" refers to how the elements of the set are organized, and even for simple familiar objects, the topologies are very subtle.

Angles are illustrative of this phenomenon. Since the set of angles "closes up" —- when you go around a full 360^o — the set cannot be identified with a set of numbers or vectors in a completely satisfactory way. One has to introduce special cases to handle exceptions, and there is no way to get around this.

Other situations, like sets of lines in the plane or 3-space, lead to even more complicated topologies. For these two cases, the set is *nonorientable,* like a Möbius band, and this is particularly difficult to coordinatize.

Projective geometry began by the efforts of Renaissance architects and artists to deal with perspective. *Projective space* enlarges our usual space by introducing new points (called *ideal points*) which is where parallel lines eventually meet. (Imagine an aerial view of railroad tracks converging in the horizon.)

Projective space also has a very complicated topology; for example, the projective plane is nonorientable. It enjoys a set of *homogeneous coordinates,* which are only unique up to scaling — and to do calculations in projective geometry one has to work only in pieces of the space which are manageable and do admit vector coordinates. The geometric calculations all reduce to matrix operations in linear algebra, but to specify an arbitrary point in two-dimensional projective space, one needs *three* coordinates.

Transformations and data types

Certain types of transformations work better in certain coordinate systems. For example, polar coordinates behave very well under rotations, but they can be extraordinarily awkward in others. Try writing down the expression for a translation in polar coordinates and compare it to the expression in rectilinear coordinates.

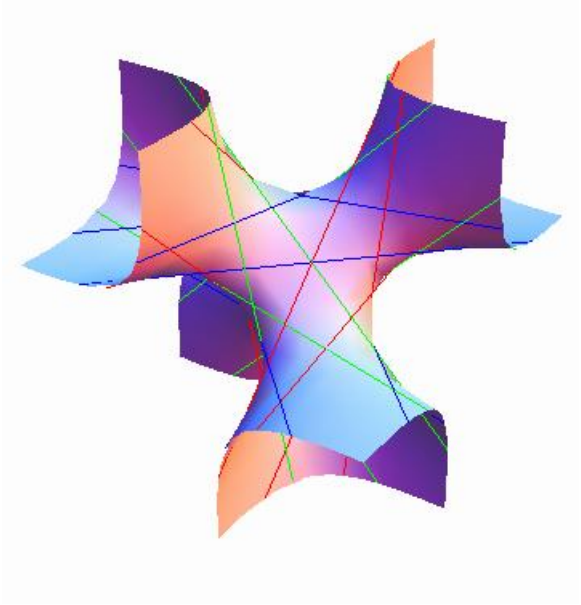
Transformations preserving some special geometric properties may be suitable for special data types, which can by more succinct and more efficient. For example, angle-preserving transformations can be written very elegantly in terms of complex numbers: if $z \in \mathbb{C}$ is a complex

number, then the *affine transformation* $z \longmapsto \lambda z + \tau$, where $\lambda \in \mathbb{C} \setminus \{0\}$ and $\tau \in \mathbb{C}$ is the most general orientation-preserving transformation which preserves angles. Whereas general affine transformations of the plane need six numbers, angle-preserving transformations depend only on two *complex numbers,* which is equivalent to four (real) numbers. This represents a significant improvement in the storage of data.

In 3 dimensions, rotations are more complicated, but they can be represented very elegantly using *quaternions,* the four-dimensional generalization of complex numbers. A linear rotation of $\mathbb{R}^3$ admits a very nice description in terms of quaternions, generalizing the remarkable formula

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$

Quaternions are more complicated than complex numbers, largely due to the fact that $AB \neq BA$ in general. However, they are easily implemented in terms of standard vector operations. Discovered in 1843 by W. R. Hamilton (long before the advent of computer graphics) they are now a standard component of graphical routines implemented in hardware.



Department of Mathematics, University of Maryland, College Park, MD 20742 USA