Computer Graphics beyond the Third Dimension:

Geometry, Orientation Control, and Rendering for Graphics in Dimensions Greater than Three

Course Notes for SIGGRAPH '98

Course Organizer

Andrew J. Hanson
Computer Science Department
Indiana University
Bloomington, IN 47405 USA
Email: hanson@cs.indiana.edu

Abstract

This tutorial provides an intuitive connection between many standard 3D geometric concepts used in computer graphics and their higher-dimensional counterparts. We begin by answering frequently asked geometric questions whose resolution, though obvious in hindsight, may be obscure to those who have never ventured beyond the third dimension. We then develop methods for describing, transforming, interacting with, and displaying geometry in arbitrary dimensions. We discuss several examples directly relevant to ordinary 3D graphics, including the treatment of quaternion frames as 4D geometric objects, and the application of generalized lighting models to 4D representations of 3D volumetric density data.

Presenter's Biography

Andrew J. Hanson is a professor of computer science at Indiana University, and has regularly taught courses in computer graphics, computer vision, programming languages, and scientific visualization. He received a BA in chemistry and physics from Harvard College in 1966 and a PhD in theoretical physics from MIT in 1971. Before coming to Indiana University, he did research in theoretical physics at the Institute for Advanced Study, Stanford, and Berkeley, and then in computer vision at the SRI Artificial Intelligence Center. He has published a variety of technical articles on machine vision, computer graphics, and visualization methods. He has also contributed three articles to the Graphics Gems series dealing with user interfaces for rotations and with techniques of N-dimensional geometry. His current research interests include scientific visualization (with applications in mathematics and physics), machine vision, computer graphics, perception, and the design of interactive user interfaces for virtual reality and visualization applications.

Contents

General Information on the Tutorial

Course Syllabus

Summary: This tutorial will bridge the gap between the familiar geometric methods of 3D computer graphics and their generalizations to higher dimensions. Participants will learn techniques for describing, transforming, interacting with, and displaying geometric objects in dimensions greater than three. Examples with direct relevance to graphics will include quaternion geometry and 3D scalar fields viewed as 4D elevation maps.

Prerequisites: Participants should be comfortable with and have an appreciation for conventional mathematical methods of 3D computer graphics used in polygon rendering, ray-tracing, and illumination models. Some knowledge of volume rendering would be helpful. These methods will form the assumed common language upon which the tutorial is based.

Objectives: Participants will learn basic methods of high-dimensional geometry as intuitive generalizations of 3D graphics techniques. Specific concepts such as projection, normal determination, and generalized lighting will be applied to examples such as the visualization of quaternions and 3D scalar fields.

Outline: This is a two-hour tutorial and the material will be arranged approximately as follows:

I. **Introduction to N-dimensional geometry.**

   A. *(45 min)* Develop formulas and techniques of N-dimensional geometry as generalizations that clarify standard 3D geometric methods such as normals, cross-products, inside-outside tests, proximity calculations, and barycentric coordinates. Approaches to treating N-dimensional data points as geometry.

   B. *(25 min)* Rotations in N dimensions and natural interfaces for N-dimensional orientation control. Quaternion representations of orientation.

II. **Applications in low dimensions.**

   A. *(25 min)* Moving 3D frames as 4D quaternion differential equations; the Frenet and Bishop frame equations; visualizing quaternion frames as 4D geometric objects.

   B. *(25 min)* Displaying 4D geometric objects. Example: viewing 3D scalar fields as 4D elevation maps. Approaches to display, virtual lighting, and interaction with $D >= 4$ objects in 3D virtual reality environments.

1 Overview

Practitioners of photorealistic computer graphics use many mathematical tools that have origins in linear algebra, analytic and differential geometry, and the geometric constructs of classical and even quantum physics. Another growing specialty in graphics is the representation, interactive manipulation, and display of multi-dimensional data. However, the relationships between the methods used by these two graphics applications are not widely understood or exploited. The purpose of this Tutorial is to construct an intuitive bridge between many standard 3D geometric concepts and their higher-dimensional counterparts.

The Tutorial will begin with the answers to frequently asked geometric questions whose form, though obvious in hindsight, may be obscure to those who have never ventured beyond the third dimension. By exploring and generalizing core techniques used throughout ray-tracing, animation, and photorealistic applications, we will expose powerful arbitrary-dimensional concepts that help establish geometric interpretations and methods applicable to multi-dimensional data. At the same time, we will point out how arbitrary-dimensional generalizations of geometry provide new insights into familiar 3D problem domains. Finally, we will briefly note a number of very useful special cases that occur in "low" dimensions, particularly in 4D, that are directly relevant to ordinary 3D graphics; these topics will include simple intuitions about quaternions as 4D geometric objects, the multifaceted relation of moving quaternions to time-dependent 3D orientation frames, and the generalization of classical lighting models to create intuitive 4D representations of 3D volumetric density data.

2 Fundamental Concepts

In computer graphics, we begin with a certain family of geometric ideas and mathematical implementations that allow us to generate images by simulating the physical interaction of materials and light. These ideas are taught for the most part with very little emphasis on their generality, and many of the terms and techniques are very specific to three dimensions. The specificity to 3D is of course reasonable, since that is the exclusive target of photorealistic graphics problems. However, there are fascinating issues involved in generalizing the computations involved to higher dimensions: for one thing, the intrinsic nature of the geometry is sometimes clarified by working in a general dimension and then reducing it back to 3D to clarify which properties are general and which are specific to 3D; for another, there are numerous classes of data relevant to graphics, including, for example, quaternion representations of rotations, that are intrinsically of dimension greater than three. Other areas such as those involving N-dimensional data points from statistical surveys and such are relevant to this tutorial in a peripheral sense: while we are concerned mainly with *geometry*, that is, the *connections* among points to create curves, surfaces, and higher-dimensional manifolds, there will be certain parts such as rotation control interfaces that are also useful for projections of high-dimensional point sets lacking any other geometry.

The basic material covered in these notes will begin with notions that allow one to easily discuss geometry in many dimensions from a unified viewpoint, and then we will keep dropping back to 3D to see where the familiar specializations come from.

3 Geometry for N-Dimensional Graphics

(This section of the lecture notes closely follows the author's Graphics Gems IV article [37].)

Textbook graphics treatments commonly use special notations for the geometry of 2 and 3 dimensions that are not obviously generalizable to higher dimensions. Our purpose here will be to study a family of geometric formulas frequently used in graphics that are easily extendible to N dimensions as well as being helpful alternatives to standard 2D and 3D notations.

What use are such formulas? In mathematical visualization, which commonly must deal with higher dimensions — 4 real dimensions, 2 complex dimensions, etc. — the utility is self-evident (see, e.g., [5, 31, 44, 65]). The visualization of statistical data also frequently utilizes techniques of N-dimensional display (see, e.g., [64, 25, 26, 42]). We hope that maing some of the basic techniques more available will encourage further exploitation of N-dimensional graphics in scientific visualization problems.

We classify the formulas we will study into the following categories: basic notation and the N-simplex; rotation formulas; imaging in N-dimensions; N-dimensional hyperplanes and volumes; N-dimensional cross-products and normals; clipping formulas; the point-hyperplane distance; barycentric coordinates and parametric hyperplanes; N-dimensional ray-tracing methods. An appendix collects a set of obscure Levi-Civita symbol techniques for computing with determinants. For additional details and insights, we refer the reader to classic sources such as [75, 18, 52, 3, 22].

4 Rotations for N-Dimensional Graphics

(This section of the lecture notes closely follows the author's Graphics Gems V article [38].)

In the previous article [37], "Geometry for N-Dimensional Graphics," we described a family of techniques for dealing with the geometry of N-dimensional models in the context of graphics applications. In this section, we build on that framework to look in more detail at rotations in N-dimensional Euclidean space. In particular, we give a natural N-dimensional extension of the 3D rolling ball technique described in Graphics Gems III [35], "The Rolling Ball," along with the corresponding analog of the Virtual Sphere method [16]. Next, we touch on practical methods for specifying and understanding the parameters of N-dimensional rotations. Finally, we give the explicit 4D extension of 3D quaternion orientation splines.

5 Quaternion Frames

In this section of the lecture notes, we study the nature of quaternions, which are effectively points on a 4D object, the three-sphere; the three-sphere (S^3) is analogous to an ordinary ball or two-sphere (S^2) embedded in 3D, except that the three-sphere is a solid object instead of a surface. To manipulate, display, and visualize rotations in 3D, we may convert 3D rotations to 4D quaternion points and treat the entire problem in the framework of 4D geometry. The methods in this section follow closely techniques introduced in Hanson and Ma [45, 46] for representing families of coordinate frames on curves in 3D as curves in the 4D quaternion space. The extension to coordinate

frames on surfaces and the corresponding induced surfaces in quaternion space are introduced in "Constrained Optimal Framings of Curves and Surfaces using Quaternion Gauss Maps" [39].

6 Displaying and Lighting 4D Objects

A natural testing ground for the concept of graphics in higher dimensions is to consider volumetric objects in 4D: then we imitate the usual projection of surface patches from 3D to 2D film by projecting volumetric patches from 4D to "3D film." This has been worked out for a number of natural application domains in mathematics [41, 42, 40, 19, 6, 8].

A visual introduction to the techniques for rendering 4D solid objects projected to a 3D imaging space is available in the video animation, "FourSight," available in *Siggraph Video Review* volume 85 [43].

While these techniques may seem a little esoteric, in fact they should be quite familiar: any volume-renderable 3D scalar field such as a CT scan, MR image, or 3D pressure density is indistinguishable mathematically from a 4D elevation map [42]. To make the analogy clear, consider a continuous single-valued function $z = f(x, y)$ giving, say, the elevation z at each point (x, y) on a small patch of the earth. We can represent this graphically in many ways, but one of the options is to create an alternative 2D image in which we view the data obliquely as a smoothly varying 3D mountain range lit by the sun. A 3D scalar field is just a similar function $w = f(x, y, z)$, and thus can be analogously viewed as an alternative 3D volume rendering created by a oblique 4D view with 4D lighting.

Going beyond four dimensions is of course much harder. Nevertheless, the mathematical methods involved in trying to create images of various sorts of higher-dimensional data should now be relatively straightforward. Our readers should now be able to deal with many low-level mathematical problems presented by the manipulation of geometry in dimensions greater than three, and to focus their energies on the much more difficult question of trying to figure out which display methods communicate such material effectively to the human viewer.

Acknowledgments

The slightly edited versions of the author's papers from Graphics Gems IV and V [37, 38] are included in the CDROM and Course Notes with the kind permission of Academic Press. The republication in the Course Notes of three key papers from IEEE Transactions on Visualization and Computer Graphics [46], IEEE Computer Graphics and Applications [44], and from the Proceedings of IEEE Visualization [42] appear with permission of the IEEE Computer Society Press.

References

[1] B. Alpern, L. Carter, M. Grayson, and C. Pelkie. Orientation maps: Techniques for visualizing rotations (a consumer's guide). In *Proceedings of Visualization '93*, pages 183–188. IEEE Computer Society Press, 1993.

[2] S. L. Altmann. *Rotations, Quaternions, and Double Groups*. Oxford University Press, 1986.

[3] T. Banchoff and J. Werner. *Linear Algebra through Geometry*. Springer-Verlag, 1983.

[4] T. F. Banchoff. Visualizing two-dimensional phenomena in four-dimensional space: A computer graphics approach. In E. Wegman and D. Priest, editors, *Statistical Image Processing and Computer Graphics*, pages 187–202. Marcel Dekker, Inc., New York, 1986.

[5] Thomas F. Banchoff. *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions*. Scientific American Library, New York, NY, 1990.

[6] David Banks. Interactive display and manipulation of two-dimensional surfaces in four dimensional space. In *Symposium on Interactive 3D Graphics*, pages 197–207, New York, 1992. ACM.

[7] David Banks. Interactive manipulation and display of two-dimensional surfaces in four-dimensional space. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 197–207, March 1992.

[8] David C. Banks. Illumination in diverse codimensions. In *Computer Graphics*, pages 327–334, New York, 1994. ACM. Proceedings of SIGGRAPH 1994; Annual Conference Series 1994.

[9] A. Barr, B. Currin, S. Gabriel, and J. Hughes. Smooth interpolation of orientations with angular velocity constraints using quaternions. In *Computer Graphics Proceedings, Annual Conference Series*, pages 313–320, 1992. Proceedings of SIGGRAPH '92.

[10] Richard L. Bishop. There is more than one way to frame a curve. *Amer. Math. Monthly*, 82(3):246–251, March 1975.

[11] Wilhelm Blaschke. *Kinematik und Quaternionen*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1960.

[12] Jules Bloomenthal. Calculation of reference frames along a space curve. In Andrew Glassner, editor, *Graphics Gems*, pages 567–571. Academic Press, Cambridge, MA, 1990.

[13] Kenneth A. Brakke. The surface evolver. *Experimental Mathematics*, 1(2):141–165, 1992. The "Evolver" system, manual, and sample data files are available by anonymous ftp from geom.umn.edu, The Geometry Center, Minneapolis MN.

[14] D. W. Brisson, editor. *Hypergraphics: Visualizing Complex Relationships in Art, Science and Technology*, volume 24. Westview Press, 1978.

[15] S. A. Carey, R. P. Burton, and D. M. Campbell. Shades of a higher dimension. *Computer Graphics World*, pages 93–94, October 1987.

[16] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-d rotation using 2-d control devices. In *Proceedings of Siggraph 88*, volume 22, pages 121–130, 1988.

[17] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-d rotation using 2-d control devices. In *Computer Graphics*, volume 22, pages 121–130, 1988. Proceedings of SIGGRAPH 1988.

[18] H.S.M. Coxeter. *Regular Complex Polytopes*. Cambridge University Press, second edition, 1991.

[19] R. A. Cross and A. J. Hanson. Virtual reality performance for virtual geometry. In *Proceedings of Visualization '94*, pages 156–163. IEEE Computer Society Press, 1994.

[20] A. R. Edmonds. *Angular Momentum in Quantum Mechanics*. Princeton University Press, Princeton, New Jersey, 1957.

[21] A.R. Edmonds. *Angular Momentum in Quantum Mechanics*. Princeton University Press, Princeton, New Jersey, 1957.

[22] N.V. Efimov and E.R. Rozendorn. *Linear Algebra and Multi-Dimensional Geometry*. Mir Publishers, Moscow, 1975.

[23] T. Eguchi, P. B. Gilkey, and A. J. Hanson. Gravitation, gauge theories and differential geometry. *Physics Reports*, 66(6):213–393, December 1980.

[24] L. P. Eisenhart. *A Treatise on the Differential Geometry of Curves and Surfaces*. Dover, New York, 1909 (1960).

[25] S. Feiner and C. Beshers. Visualizing n-dimensional virtual worlds with n-vision. *Computer Graphics*, 24(2):37–38, March 1990.

[26] S. Feiner and C. Beshers. Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds. In *Proceedings of UIST '90, Snowbird, Utah*, pages 76–83, October 1990.

[27] Gerd Fischer. *Mathematische Modelle*, volume I and II. Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden, 1986.

[28] H. Flanders. *Differential Forms*. Academic Press, New York, 1963.

[29] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, second edition, 1990. page 227.

[30] A. R. Forsyth. *Geometry of Four Dimensions*. Cambridge University Press, 1930.

[31] George K. Francis. *A Topological Picturebook*. Springer Verlag, 1987.

[32] Herbert Goldstein. *Classical Mechanics*. Addison-Wesley, 1950.

[33] Alfred Gray. *Modern Differential Geometry of Curves and Surfaces*. CRC Press, Inc., Boca Raton, FL, second edition, 1998.

[34] Cindy M. Grimm and John F. Hughes. Modeling surfaces with arbitrary topology using manifolds. In *Computer Graphics Proceedings, Annual Conference Series*, pages 359–368, 1995. Proceedings of SIGGRAPH '95.

[35] A. J. Hanson. The rolling ball. In David Kirk, editor, *Graphics Gems III*, pages 51–60. Academic Press, Cambridge, MA, 1992.

[36] A. J. Hanson. A construction for computer visualization of certain complex curves. *Notices of the Amer.Math.Soc.*, 41(9):1156–1163, November/December 1994.

[37] A. J. Hanson. Geometry for n-dimensional graphics. In Paul Heckbert, editor, *Graphics Gems IV*, pages 149–170. Academic Press, Cambridge, MA, 1994.

[38] A. J. Hanson. Rotations for n-dimensional graphics. In Alan Paeth, editor, *Graphics Gems V*, pages 55–64. Academic Press, Cambridge, MA, 1995.

[39] A. J. Hanson. Constrained optimal framings of curves and surfaces using quaternion Gauss maps, 1998. Submitted for publication.

[40] A. J. Hanson and R. A. Cross. Interactive visualization methods for four dimensions. In *Proceedings of Visualization '93*, pages 196–203. IEEE Computer Society Press, 1993.

[41] A. J. Hanson and P. A. Heng. Visualizing the fourth dimension using geometry and light. In *Proceedings of Visualization '91*, pages 321–328. IEEE Computer Society Press, 1991.

[42] A. J. Hanson and P. A. Heng. Four-dimensional views of 3d scalar fields. In *Proceedings of Visualization '92*, pages 84–91. IEEE Computer Society Press, 1992.

[43] A. J. Hanson and P. A. Heng. Foursight. In *Siggraph Video Review*, volume 85. ACM Siggraph, 1992. Scene 11, Presented in the Animation Screening Room at SIGGRAPH '92, Chicago, Illinois, July 28–31, 1992.

[44] A. J. Hanson and P. A. Heng. Illuminating the fourth dimension. *Computer Graphics and Applications*, 12(4):54–62, July 1992.

[45] A. J. Hanson and H. Ma. Visualizing flow with quaternion frames. In *Proceedings of Visualization '94*, pages 108–115. IEEE Computer Society Press, 1994.

[46] A. J. Hanson and H. Ma. Quaternion frame approach to streamline visualization. *IEEE Trans. on Visualiz. and Comp. Graphics*, 1(2):164–174, June 1995.

[47] A. J. Hanson and H. Ma. Space walking. In *Proceedings of Visualization '95*, pages 126–133. IEEE Computer Society Press, 1995.

[48] A. J. Hanson, T. Munzner, and G. K. Francis. Interactive methods for visualizable geometry. *IEEE Computer*, 27(7):73–83, July 1994.

[49] Andrew J. Hanson, Pheng A. Heng, and Brian C. Kaplan. Techniques for visualizing Fermat's last theorem: A case study. In *Proceedings of Visualization 90*, pages 97–106, San Francisco, October 1990. IEEE Computer Society Press.

[50] John C. Hart, George K. Francis, and Louis H. Kauffman. Visualizing quaternion rotation. *ACM Trans. on Graphics*, 13(3):256–276, 1994.

[51] D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination*. Chelsea, New York, 1952.

[52] John G. Hocking and Gail S. Young. *Topology*. Addison-Wesley, 1961.

[53] C. Hoffmann and J. Zhou. Some techniques for visualizing surfaces in four-dimensional space. *Computer-Aided Design*, 23:83–91, 1991.

[54] S. Hollasch. Four-space visualization of 4D objects. Master's thesis, Arizona State University, August 1991.

[55] B. Jüttler. Visualization of moving objects using dual quaternion curves. *Computers and Graphics*, 18(3):315–326, 1994.

[56] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 271–280, July 1989.

[57] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Computer Graphics Proceedings, Annual Conference Series*, pages 369–376, 1995. Proceedings of SIGGRAPH '95.

[58] F.. Klock. Two moving coordinate frames for sweeping along a 3d trajectory. *Computer Aided Geometric Design*, 3, 1986.

[59] H. Ma. *Curve and Surface Framing for Scientific Visualization and Domain Dependent Navigation*. PhD thesis, Indiana University, February 1996.

[60] H. Ma and A. J. Hanson. Meshview. A portable 4D geometry viewer written in OpenGL/Motif, available by anonymous ftp from geom.umn.edu, The Geometry Center, Minneapolis MN.

[61] J. Milnor. *Topology from the Differentiable Viewpoint*. The University Press of Virginia, Charlottesville, 1965.

[62] Hans Robert Müller. *Sphärische Kinematik*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1962.

[63] G. M. Nielson. Smooth interpolation of orientations. In N.M. Thalman and D. Thalman, editors, *Computer Animation '93*, pages 75–93, Tokyo, June 1993. Springer-Verlag.

[64] Michael A. Noll. A computer technique for displaying n-dimensional hyperobjects. *Communications of the ACM*, 10(8):469–473, August 1967.

[65] Mark Phillips, Silvio Levy, and Tamara Munzner. Geomview: An interactive geometry viewer. *Notices of the Amer. Math. Society*, 40(8):985–988, October 1993. Available by anonymous ftp from geom.umn.edu, The Geometry Center, Minneapolis MN.

[66] D. Pletincks. Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, 5(1):2–13, 1989.

[67] Ravi Ramamoorthi and Alan H. Barr. Fast construction of accurate quaternion splines. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 287–292. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[68] John Schlag. Using geometric constructions to interpolate orientation with quaternions. In James Arvo, editor, *Graphics Gems II*, pages 377–380. Academic Press, 1991.

[69] Uri Shani and Dana H. Ballard. Splines as embeddings for generalized cylinders. *Computer Vision, Graphics, and Image Processing*, 27:129–156, 1984.

[70] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 63–70, November 1990.

[71] K. Shoemake. Animating rotation with quaternion curves. In *Computer Graphics*, volume 19, pages 245–254, 1985. Proceedings of SIGGRAPH 1985.

[72] K. Shoemake. Animation with quaternions. Siggraph Course Lecture Notes, 1987.

[73] Ken Shoemake. Arcball rotation control. In Paul Heckbert, editor, *Graphics Gems IV*, pages 175–192. Academic Press, 1994.

[74] Ken Shoemake. Fiber bundle twist reduction. In Paul Heckbert, editor, *Graphics Gems IV*, pages 230–236. Academic Press, 1994.

[75] D.M.Y. Sommerville. *An Introduction to the Geometry of N Dimensions*. Reprinted by Dover Press, 1958.

[76] N. Steenrod. *The Topology of Fibre Bundles*. Princeton University Press, 1951. Princeton Mathematical Series 14.

[77] K. V. Steiner and R. P. Burton. Hidden volumes: The 4th dimension. *Computer Graphics World*, pages 71–74, February 1987.

[78] D. J. Struik. *Lectures on Classical Differential Geometry*. Addison-Wesley, 1961.

[79] P.G. Tait. *An Elementary Treatise on Quaternions*. Cambridge University Press, 1890.

[80] J. R. Weeks. *The Shape of Space*. Marcel Dekker, New York, 1985.

[81] S. Weinberg. *Gravitation and Cosmology: Principles and Applications of General Relativity*. John Wiley and Sons, 1972.

[82] E.T. Whittaker. *A Treatise on the Analytical Dynamics of Particles and Rigid Bodies*. Dover, New York, New York, 1944.

Computer Graphics

beyond the Third Dimension

Andrew J. Hanson

Computer Science Department

Indiana University

SIGGRAPH '98 TUTORIAL

Outline

- **Part I(A): N-Dimensional Geometry for Computer Graphics**
- **Part I(B): N-Dimensional Rotations for Computer Graphics**
- **Part II(A): Quaternion Frames**
- **Part II(B): Four-Dimensional Rendering and 3D Scalar Fields**

Methodology

- **Background:** Knowledge of geometric concepts underlying 3D graphics: transformations, projections, rays, proximity, inside-outside tests, normals, lighting and shading heuristics, volume rendering.
- **What we will do:** Start from 3D graphics:

  Graphics = simulation of the physics of light interacting with matter in 3D

  and imagine we live in a different world:

  ND Graphics = simulation of the physics of *hypothetical* light interacting with *hypothetical* matter in ND
- **Practical 4D applications:** Quaternion frame fields; 3D data as mountains on 4D terrain.

Key Concepts

- **Vectors:** A list of N numbers; homogeneous form for projection uses $(N+1)$ numbers.
- **Matrix Algebra:** Vectors are acted on by $N \times N$ matrix multiplication to perform needed operations such as rotations.
- **Determinants:** Almost every extension from *ungeneralizable* 3D graphics geometry to ND graphics geometry makes use of determinants.
- **Polytopes:** A systematic extension of geometry from dimension to dimension: (point, line segment, triangle, tetrahedron, ...)

FINAL SUMMARY

Let's pick four major ideas to take home:

- **Cross Product in ND** is a determinant of vectors with empty last column, *direction* is the normal to hyperface, *size* gives face's $(N-1)$-volume.
- **Rolling Ball Rotations** in ND take $(N-1)$-dim tan vector as input, but adding $(N-1)(N-2)/2$ subplanes of motion gives *full* $N(N-1)/2$ rotational degrees of freedom.
- **Quaternions** generalize half-angle formula for 2D rotations to 3D, obey a "square-root-like" angular differential equation, become 4D curves describing twisting of ordinary 3D curve.
- **4D Light** creates volume-rendered shadings that *should* be just as interpretable as a newspaper photo, and you can render 3D scalar fields this way as 4D mountain ranges.

Part I(A): N-Dimensional Geometry for Computer Graphics

Andrew J. Hanson

Computer Science Department

Indiana University

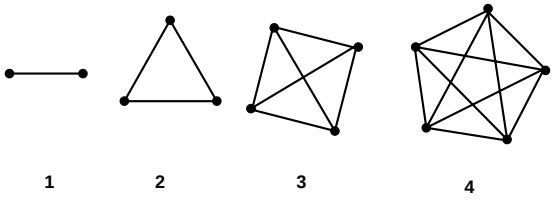
Outline

- **The Simplex:** A simple basis for N-Dim objects
- **Cross Product:** *Frequently Asked!*
- **Ray casting and projection**
- **Hyperplanes**
- **Barycentric coordinates, clipping, etc.**
- **Volumes and subvolumes**

Motivation

- **Usual 3D graphics treatments of geometry are customized to 3D.**
- **General-dimension formulas** aren't really harder, but give additional insight.
- **N dimensional formulas for:** normals, cross products, ray-tracing, barycentric coordinates, etc., are essential for high-dimensional applications.

Basic Ideas: the simplex



2D projections of simplexes with dimension 1–4.

Basic Ideas: the simplex

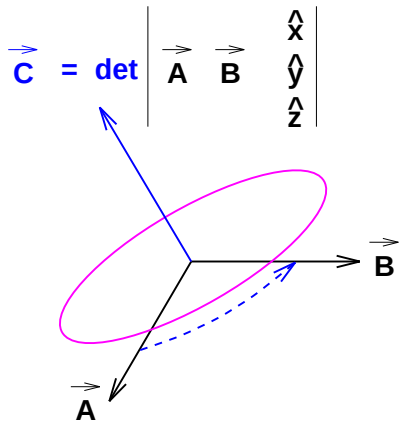
- Simplex generalizes line segment, triangle, etc., to ND.
- N-simplex has $(N+1)$ points.
- Standard coordinates: $\{(0,0,\ldots,0),\ (1,0,\ldots,0),\ (0,1,0,\ldots,0),\ldots,\ (0,0,\ldots,0,1)\}$.
- $(N+1)$ *linearly independent* points of simplex define a *hyperplane*.
- *Last vector* in standard coordinates defines positive direction of oriented $(N-1)$ subspace.

MOST FREQUENTLY ASKED QUESTION

. . . turns out to be this:

What is the 4D cross product?

This picture of the 3D cross product is well known:



Analog for higher dimensions is confusing . . .

Finding the 4D cross product . . .

- **3D cross product:** intuition doesn't generalize.
- Rotation of two vectors leaves *plane*, not *vector* fixed.
- **Solution:** Analog of right-hand rule comes from Simplex:

  $\Rightarrow$ Need $(N-1)$ vectors, not 2, to make Cross Product.

Generalize Cross Product

Correct generalization comes from these simple concepts:

- **Preserve Simplex Hierarchy.** Dot of N-th basis vector with $(N-1)$-simplex's cross product should give positive value in each dimension.
- **Relate ND Cross Product to measure:** magnitude should be related to $(N-1)$ volume (e.g., *area* in 3D).
- **Relate to N-volume:** N-th basis vector dotted with Cross Product should give N-volume.
- **Cross product *defines* normal vector.** This is how we determine signed hyperface normals by analogy to 3D.

Cross Product Equations

This definition obeys the given rules (violated by many people, including Mathematica and me in my early papers!):

- **Given:** ordered set of $(N-1)$ edge vectors $(\vec{x}_k - \vec{x}_0)$: (the edges of one of the $(N-1)$-simplexes in an object's tessellation.)
- **Normal Vector** is then the *generalized cross-product* whose components are cofactors of the last column in the following (notationally abusive!) determinant:

$$\vec{n} = n_x\hat{\mathbf{x}} + n_y\hat{\mathbf{y}} + n_z\hat{\mathbf{z}} + \cdots + n_w\hat{\mathbf{w}} =$$
$$\det\begin{bmatrix} (x_1-x_0) & (x_2-x_0) & \cdots & (x_{N-1}-x_0) & \hat{\mathbf{x}} \\ (y_1-y_0) & (y_2-y_0) & \cdots & (y_{N-1}-y_0) & \hat{\mathbf{y}} \\ (z_1-z_0) & (z_2-z_0) & \cdots & (z_{N-1}-z_0) & \hat{\mathbf{z}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (w_1-w_0) & (w_2-w_0) & \cdots & (w_{N-1}-w_0) & \hat{\mathbf{w}} \end{bmatrix}$$

Magnitude of Cross Products

We note a remarkable combinatorial formula, the generalization to ND of the 3D formula

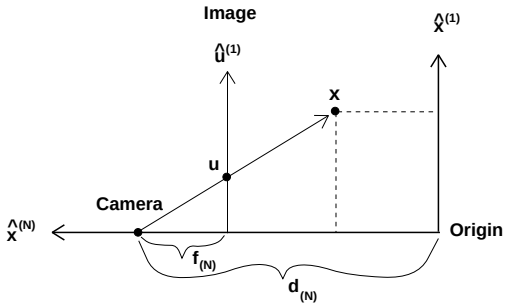
$$(\vec{A}\times\vec{B})\cdot(\vec{A}\times\vec{B}) = (\vec{A}\cdot\vec{A})(\vec{B}\cdot\vec{B}) - (\vec{A}\cdot\vec{B})^2$$

The *magnitude* as well as the *direction* of a normal is important:

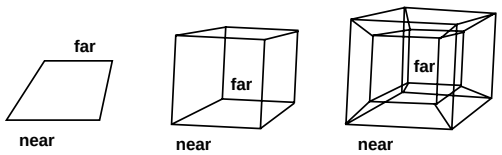
$$\vec{n}\cdot\vec{n} =$$
$$\det\begin{bmatrix} v(1,1) & v(1,2) & \cdots & v(1,N-1) \\ v(2,1) & v(2,2) & \cdots & v(2,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ v(N-1,1) & v(N-1,2) & \cdots & v(N-1,N-1) \end{bmatrix}$$
$$= \left((N-1)!\,V_{N-1}\right)^2 .$$

Here $v(i,j) = (\vec{x}_i - \vec{x}_0)\cdot(\vec{x}_j - \vec{x}_0)$, and V_{N-1} is the volume of the $(N-1)$ simplex (more later).

Rays and Perspective



Projection process for an N-dimensional pinhole camera.

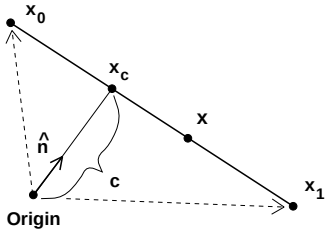


Perspective projections of a wire-frame square, a cube, and a hypercube in 2D, 3D, and 4D, respectively.

ND Projection Procedures

- **Orthographic Projections:** Throw out the N-th coordinate $x^{(N)}$ of each point.
- **3D example:**
  Project (x, y, z) to $(xf/(d-z), yf/(d-z))$.
- **ND Perspective Scaling.** Divide first $(N-1)$ coordinates by $(d_N - x^{(N)})/f_N$
- **Repeat Recursively** down to 2D image. Hierarchy of up to $(N-2)$ parameter sets $\{(f_N, d_N), \ldots, (f_3, d_3)\}$ may be used.

Relate Hyperplane to Simplex



The 2D line from $\vec{x}_0$ to $\vec{x}_1$ obeying the equation $\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = 0$. The constant c is just $\hat{\mathbf{n}} \cdot \vec{x}_0$

Generalization to ND requires N points $(\vec{x}_0, \ldots, \vec{x}_{N-1}$ and their normalized cross-product $\hat{\mathbf{n}}$:

- **Hyperplane through $\vec{x}_0$ perpendicular to $\hat{\mathbf{n}}$:**

$$\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = 0$$

- **Point closest to origin:** $\vec{x}_c = c\hat{\mathbf{n}}$.

Volumes and subvolumes

The *volume* of an N-simplex is the determinant of its $(N+1)$ defining points ($\approx$ magnitude of its normal in $(N+2)$ dimensions!):

$$V_N = \frac{1}{N!} \det \begin{bmatrix} x_1 & x_2 & \cdots & x_N & x_0 \\ y_1 & y_2 & \cdots & y_N & y_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_1 & w_2 & \cdots & w_N & w_0 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} .$$

- **Bottom row of 1's:** homogeneous coordinates.
- **This is a SIGNED volume:** implicitly defines the N-dimensional generalization of the *Right-Hand Rule*.
- **Disastrous sign inconsistencies** unless $\vec{x}_0$ is in the last column as shown (typically $\vec{x}_0 = (0, 0, \ldots, 0, 0; 1)$) .

Tricks for volumes

In 3D, cross-product gives volume of parallelepiped:

$$[(\vec{x}_1 - \vec{x}_0) \times (\vec{x}_2 - \vec{x}_0)] \cdot (\vec{x}_3 - \vec{x}_0) ,$$

Tetrahedron with vertices at the points $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3)$ has *one-sixth* the volume of the parallelepiped.

N dimensional simplex volume is $1/N!$ times the volume of the parallelepiped whose edges are given by the matrix columns.

Invariant Subspace Forms

Properties of determinants give *squared volume* as:

$$(V_N)^2 = \left(\frac{1}{N!}\right)^2 \det |X^t \cdot X|$$

$$= \left(\frac{1}{N!}\right)^2 \det \begin{bmatrix} v(1,1) & v(1,2) & \cdots & v(1,N) \\ v(2,1) & v(2,2) & \cdots & v(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ v(N,1) & v(N,2) & \cdots & v(N,N) \end{bmatrix} ,$$

where $v(i,j) = (\vec{x}_i - \vec{x}_0) \cdot (\vec{x}_j - \vec{x}_0)$.

This formula is the *key to a trick* for volume forms of *subspaces* of N-dimensional spaces: cannot form square matrices!

Invariant Subspace Forms

- V_K, volume for $K < N$, is *not expressible* in terms of a square matrix of coordinate differences like V_N.
- **But** V_K *is* the determinant of a square matrix in one particular coordinate frame (e.g, rotate triangle to $(x, y$ plane)).
- Multiply by transpose: *frame-independent* quadratic form.
- Thus V_K is written in terms of its K basis vectors $(\vec{x}_k - \vec{x}_0)$ of dimension N as:

$$
\begin{aligned}
(V_K)^2 &= \left(\frac{1}{K!}\right)^2 \det \begin{bmatrix} \vec{x}_1 - \vec{x}_0 \\ \vec{x}_2 - \vec{x}_0 \\ \vdots \\ \vec{x}_K - \vec{x}_0 \end{bmatrix} \cdot \\
&\quad \begin{bmatrix} \vec{x}_1 - \vec{x}_0 & \vec{x}_2 - \vec{x}_0 & \cdots & \vec{x}_K - \vec{x}_0 \end{bmatrix} \\
&= \left(\frac{1}{K!}\right)^2 \det \begin{bmatrix} v(1,1) & v(1,2) & \cdots & v(1,K) \\ v(2,1) & v(2,2) & \cdots & v(2,K) \\ \vdots & \vdots & \ddots & \vdots \\ v(K,1) & v(K,2) & \cdots & v(K,K) \end{bmatrix}
\end{aligned}
$$

Summary: Invariant Subvolumes

To compute a volume of dimension K in N dimensions:

- Find the K independent basis vectors spanning the subspace.
- Form a square $K \times K$ matrix of dot products related to V_K^2 by multiplying the $N \times K$ matrix of column vectors by its transpose on the left.

When $K = 1$, we have simply the squared Euclidean distance in N dimensions
$v(1,1) = (\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0)$.

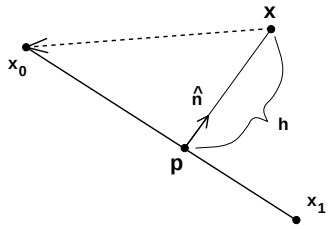
Point-Hyperplane Distance

- General formula: parallelepiped volume = base times height. But height h = distance to hyperplane.
- **Solve for** h using *ratio* of volumes:

$$h = \frac{W_N}{W_{N-1}} = \frac{N!\, V_N}{(N-1)!\, V_{N-1}} = \frac{N\, V_N}{V_{N-1}} \,.$$

Note! Here one must use the trick above to express W_{N-1} in terms of the square root of a square determinant given by the product of two non-square matrices.

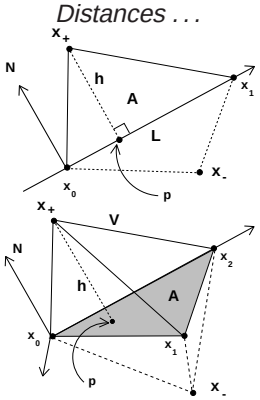
Point-Hyperplane Distance



- **Intersection Point.** Line from $\vec{x}$ to the base hyperplane along the normal $\hat{\mathbf{n}}$ to the hyperplane as $\vec{x}(t) = \vec{x} + t\hat{\mathbf{n}}$, writing the implicit equation for the hyperplane as $\hat{\mathbf{n}} \cdot (\vec{x}(t) - \vec{x}_0) = 0$, and solving for the mutual solution $t_p = \hat{\mathbf{n}} \cdot (\vec{x}_0 - \vec{x}) = -h$. Thus

$$
\begin{aligned}
\vec{p} &= \vec{x} + \hat{\mathbf{n}}(\hat{\mathbf{n}} \cdot (\vec{x}_0 - \vec{x})) \\
&= \vec{x} - h\hat{\mathbf{n}} \,.
\end{aligned}
$$

Distances . . .

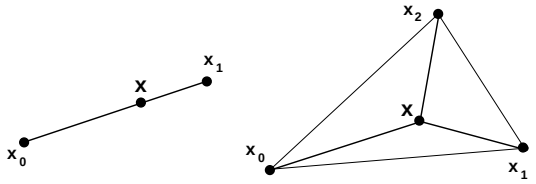


Basic idea: distances to hyperplanes *vanish*, $h = 0$, if test point is on hyperplane, i.e., volume vanishes.

Signed volumes divide plane into two regions, one with $h > 0$, the other with $h < 0$.

The pictures show how the distance h from a point to a hyperplane is computable from the ratio of the simplex volume to the lower-dimensional volume of its base, i.e., $2A/L$ or $3V/A$.

Barycentric Coords in ND



Any point maybe parameterized relative to an origin $\vec{\mathbf{x}}_0$ using the edges of an N-simplex and N parameters t_i:

$$\begin{aligned}
\vec{x}(t) &= \vec{x}_0 + t(\vec{x}_1 - \vec{x}_0) \\
\vec{x}(t_1, t_2) &= \vec{x}_0 + t_1(\vec{x}_1 - \vec{x}_0) + t_2(\vec{x}_2 - \vec{x}_0) \\
\vec{x}(t_1, t_2, t_3) &= \vec{x}_0 + t_1(\vec{x}_1 - \vec{x}_0) + t_2(\vec{x}_2 - \vec{x}_0) + \\
&\quad t_3(\vec{x}_3 - \vec{x}_0) \\
&\quad \cdots
\end{aligned}$$

Barycentric Coords in ND

- The interpolated point lies in the N-simplex provided

$$\begin{gathered}
0 \le t \le 1 \\
0 \le t_1 \le 1,\ 0 \le t_2 \le 1,\ 0 \le (1 - t_1 - t_2) \le 1 \\
0 \le t_1 \le 1,\ 0 \le t_2 \le 1,\ 0 \le t_3 \le 1, \\
0 \le (1 - t_1 - t_2 - t_3) \le 1 \\
\cdots
\end{gathered}$$

- **Values** of parameters t_i are just *ratios* of volume determinants coming from Kramer's rule:

$$t_1 = \frac{V(x, x_0, x_1)}{V(x_0, x_1, x_2)}, \quad t_2 = \frac{V(x, x_2, x_0)}{V(x_0, x_1, x_2)}$$

and similarly in higher dimensions.

Clipping Test from Signed Volumes

If the line to be clipped is given parametrically as $\vec{x}(t) = \vec{x}_a + t(\vec{x}_b - \vec{x}_a)$, where $\vec{x}_a$ and $\vec{x}_b$ are on opposite sides of the clipping hyperplane so $0 \le t \le 1$, then we simply plug $\vec{x}(t)$ into $V(\vec{x}) = 0$ and solve for t:

$$\begin{aligned}
t &= \frac{\det\left[\ \vec{x}_1 - \vec{x}_0 \quad \vec{x}_2 - \vec{x}_0 \quad \cdots \quad \vec{x}_a - \vec{x}_0\ \right]}{\det\left[\ \vec{x}_1 - \vec{x}_0 \quad \vec{x}_2 - \vec{x}_0 \quad \cdots \quad \vec{x}_a - \vec{x}_b\ \right]} \\
&= \frac{\hat{\mathbf{n}} \cdot (\vec{x}_a - \vec{x}_0)}{\hat{\mathbf{n}} \cdot (\vec{x}_a - \vec{x}_b)}\ .
\end{aligned}$$

Here $\hat{\mathbf{n}}$ is the normal to the clipping hyperplane.

Are Normals Vectors?

Almost!

- **Transform a Vector:** $x'^{(i)} = \sum_{j=1}^{N} R_{ij} x^{(j)}$
- **Transform a Normal:**

$$N'^{(i)} = \sum_{\substack{\text{all indices} \\ \text{except } i}} \epsilon_{i_1 i_2 \ldots i_{N-1} i} R_{i_1 j_1} x_1^{(j_1)} R_{i_2 j_2} x_2^{(j_2)} \cdots R_{i_{N-1} j_{N-1}} x_{N-1}^{(j_{N-1})}$$

$$= \sum_{j=1}^{N} R_{ij} N^{(j)} \det [R] \ .$$

- **This is a pseudotensor**: $\vec{N}$ is seen to behave as a vector for ordinary rotations (which have $\det [R] = 1$), but changes sign if $[R]$ contains an odd number of reflections.

Part I(B): N-Dimensional Rotations for Computer Graphics

Andrew J. Hanson

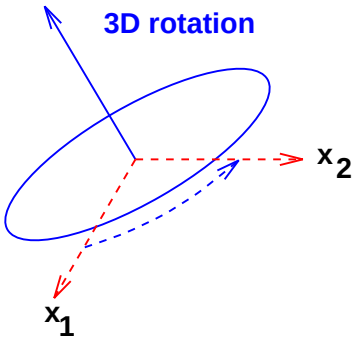
Computer Science Department

Indiana University

Outline

- **What is a rotation?** Idea of *Rotation Plane*
- **The Rolling Ball:** Context-free interaction
- **Alternate controls.**
- **Quaternions in 3D and 4D**

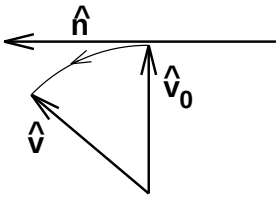
What is a Rotation?



It's NOT defined by a fixed axis!

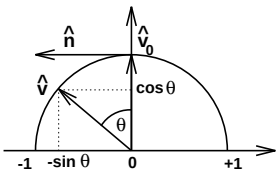
- Familiar idea of rotating "about axis" is 3D ONLY.
- Better generalization: pick two directions $\vec{x}_1$ and $\vec{x}_2$, *rotate one into the other*.
- **Remaining $(N-2)$ axes are the ND analog of familiar "fixed axis" in 3D.**

Rolling Ball Rotations



To ROLL, tilt the "north pole" vector $\hat{\mathrm{v}}_0$ in the direction of the tangent vector $\hat{\mathrm{n}}$.

Rolling Ball Rotations



Action of a right-handed 2×2 rotation M_2 in the plane of $\hat{\mathrm{v}}_0 = (0,1)$ and $\hat{\mathbf{n}} = (-1,0)$.

$$\hat{\mathrm{v}} = M_2 \cdot \hat{\mathrm{v}}_0 = \hat{\mathbf{n}} \sin\theta + \hat{\mathrm{v}}_0 \cos\theta = (-\sin\theta,\, \cos\theta)\ ,$$

Rolling Ball Rotations

The rotation matrix M_2 can be written

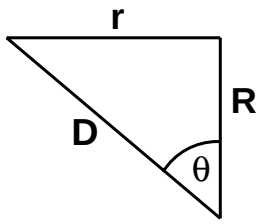
$$M_2 = \begin{bmatrix} \cos\theta & -\sin\theta \\ +\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} c & -s \\ +s & c \end{bmatrix}$$
$$= \begin{bmatrix} c & +n_x s \\ -n_x s & c \end{bmatrix}\ .$$

If we choose a right-handed overall coordinate frame, the sign of $\hat{\mathbf{n}}$ *will automatically generate the correct sign convention.*

Rolling, contd

Synopsis: Qualitatively speaking, if we imagine looking straight down at the north pole, the rolling ball *pulls* the unseen N-th component of a vector along the direction $\hat{\mathbf{n}}$ of the $(N-1)$-dimensional controller motion, bringing the unseen component gradually into view.

Rolling Rotations in ND



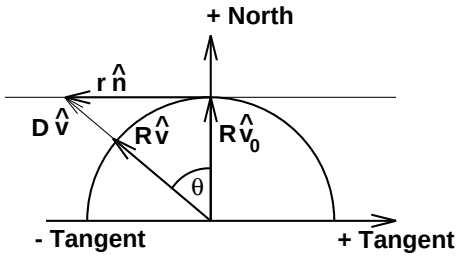
- Let R be "ball radius," rotate North vector $\hat{\mathrm{v}}_0 = (0,0,1)$ into $\hat{\mathrm{x}}$ using

$$R_0 = \begin{bmatrix} c & 0 & +s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}$$

- Here $D^2 = r^2 + R^2$, so $c = \cos\theta = R/D$ and $s = \sin\theta = r/D$.

- $\Rightarrow$ Length and direction of control vector determine *both* angle and a direction $\hat{\mathbf{n}}$.

Rolling Rotations in ND



- Already have: North vector $\hat{\mathbf{v}}_0 = (0, 0, 1)$ rotated into $\hat{\mathbf{x}}$ using $R_0(\theta)$.
- Now use *conjugation* by a second matrix R_{xy} that takes a unit $\hat{\mathbf{x}}$ vector into the completely general tangent direction $(\hat{\mathbf{n}}_x,\ \hat{\mathbf{n}}_y,\ 0)$:

$$R_{xy} = \begin{bmatrix} n_x & -n_y & 0 \\ n_y & n_x & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

to create $M_3 = R_{xy} R_0 (R_{xy})^{-1}$

Rolling Rotations in ND

The result is the 3D **Rolling Ball Matrix**:

$$M_3 = R_{xy} R_0 (R_{xy})^{-1}$$

$$= \begin{bmatrix} c + (n_y)^2(1-c) & -n_x n_y (1-c) & n_x s \\ -n_x n_y (1-c) & c + (n_x)^2 (1-c) & n_y s \\ -n_x s & -n_y s & c \end{bmatrix}$$

$$= \begin{bmatrix} 1 - (n_x)^2(1-c) & -n_x n_y (1-c) & n_x s \\ -n_x n_y (1-c) & 1 - (n_y)^2 (1-c) & n_y s \\ -n_x s & -n_y s & c \end{bmatrix}$$

4D case

The 4D case uses, e.g., **3D mouse** input $= \vec{\mathbf{r}} = (x, y, z, 0) = (r\, n_x, r\, n_y, r\, n_z, 0)$, with $n_x^2 + n_y^2 + n_z^2 = 1$.

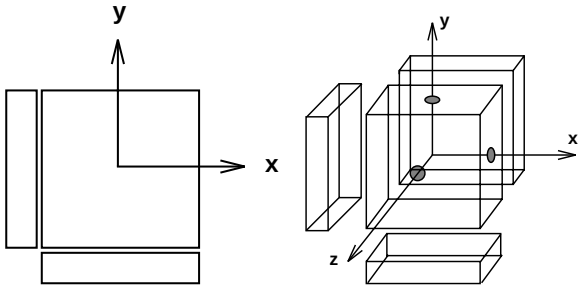
- First: transform (n_y, n_z) into a pure y-component.
- Rotate that to a pure x-component
- Rotate by θ in the (x, w)-plane, and reverse the first two rotations.

This gives

$$M_4 = R_{yz} R_{xy} R_0 (R_{xy})^{-1} (R_{yz})^{-1}$$

$$\begin{bmatrix} 1-(n_x)^2(1-c) & -(1-c)n_x n_y & -(1-c)n_x n_z & s n_x \\ -(1-c)n_x n_y & 1-(n_y)^2(1-c) & -(1-c)n_y n_z & s n_y \\ -(1-c)n_x n_z & -(1-c)n_y n_z & 1-(n_z)^2(1-c) & s n_z \\ -s n_x & -s n_y & -s n_z & c \end{bmatrix}$$

3D / 4D case



3D cube: "rolling" in x or y direction exposes hidden surfaces — the planes at $x = \pm 1$ and $y = \pm 1$.

Contrast with 4D hypercube: "rolling" a 3D mouse in the x or y or z direction exposes hidden *blocks* — the hyperplanes at $x = \pm 1$, and $y = \pm 1$, and and $z = \pm 1$.

ND case

Controller provides $(N-1)$-dimensional vector

$$\vec{\mathbf{r}} = (r\,n_1, r\,n_2, \ldots, r\,n_{N-1}, 0)$$

with $\vec{\mathbf{r}} \cdot \vec{\mathbf{r}} = r^2$ and $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$. Then with $c = \cos\theta$, $s = \sin\theta$, and $d = (1 - \cos\theta)$,

$$M_N = R_{N-2,N-1} R_{N-3,N-2} \cdots R_{1,2} R_0 (R_{1,2})^{-1} \cdots \cdots (R_{N-3,N-2})^{-1} (R_{N-2,N-1})^{-1} =$$

$$\begin{bmatrix} 1-(n_1)^2 d & -dn_2 n_1 & \cdots & -dn_{N-1} n_1 & sn_1 \\ -dn_1 n_2 & 1-d(n_2)^2 & \cdots & -dn_{N-1} n_2 & sn_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -dn_1 n_{N-1} & -dn_2 n_{N-1} & \cdots & 1-d(n_{N-1})^2 & sn_{N-1} \\ -sn_1 & -sn_2 & \cdots & -sn_{N-1} & c \end{bmatrix}$$

Summary of ND Rolling

The controller input $\vec{\mathbf{r}} = r\hat{\mathbf{n}}$ that selects the direction to "pull" can *also* determine $c = \cos\theta = R/D$, $s = \sin\theta = r/D$, with $D^2 = R^2 + r^2$, or, alternatively, $\theta = r/R$.

Then the hidden vector (like the $\hat{\mathbf{z}}$ vector in 3D) is mixed with the $\hat{\mathbf{n}}$ vector; remember, a rotation is *just a mixing of two vectors in a single plane*.

What about the rest?

We have not said how to control the rest of the orientation:

- **Full freedom** $\rightarrow N(N-1)/2$ rotational parameters (one for $N = 2$, 3 for $N = 3$, 6 for $N = 4, \ldots$).
- **Only** $N-1$ **rolling parameters** so far!
- **Commutation relation trick.** In dimensions $N > 2$, rotations are *order dependent*. If R_{ij} rotates in the plane $(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$, and the $\hat{\mathbf{x}}_N$ direction is perpendicular to the other $(N-1)$ directions, there are $(N-1)$ possible rotations $R_{iN},\ i \neq N$ parameterized by $\hat{\mathbf{n}}$.
- **Creating new rotations.** Defining $[A,\ B] = AB - BA$, one can prove the matrices R_{iN} obey

$$\begin{aligned} [R_{iN}, R_{jN}] &= \delta_{ij} R_{NN} - \delta_{jN} R_{iN} + \delta_{iN} R_{jN} - \delta_{NN} R_{ij} \\ &= -R_{ij} \end{aligned}$$

All other rotations can be derived from repeated rolling rotations!!

Columns are Axes

- **Default frame:** $\hat{\mathbf{x}}_1 = (1, 0, \ldots, 0)$, $\hat{\mathbf{x}}_2 = (0, 1, 0, \ldots, 0), \ldots, \hat{\mathbf{x}}_N = (0, \ldots, 0, 1)$
- **Desired frame:** $\hat{\mathbf{a}}_1 = (a_1^{(1)}, a_1^{(2)}, \ldots, a_1^{(N)}), \hat{\mathbf{a}}_2, \ldots, \hat{\mathbf{a}}_N$,
- **Rotation matrix** then just has the new axes as its columns:

$$M = \begin{bmatrix} \hat{\mathbf{a}}_1 & \hat{\mathbf{a}}_2 & \cdots & \hat{\mathbf{a}}_N \end{bmatrix}.$$

Concatenated subplane rotations

Rotations in the plane of a pair of coordinate axes $(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$, $i, j = 1, \ldots, N$ can be written as the block matrix

$$R_{ij}(\theta_{ij}) = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cos\theta_{ij} & 0 & \cdots & 0 & -\sin\theta_{ij} & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & \sin\theta_{ij} & 0 & \cdots & 0 & \cos\theta_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}$$

concatenation, contd

The $N(N-1)/2$ distinct $R_{ij}(\theta_{ij})$ may be concatenated in some order to produce a rotation matrix such as

$$M = \prod_{i<j} R_{ij}(\theta_{ij})$$

with $N(N-1)/2$ degrees of freedom parametrized by $\{\theta_{ij}\}$.

The matrices R_{ij} *do not commute*, so different orderings give different results.

As for 3D Euler angles, one may even repeat some matrices (with distinct parameters) and omit others, and still not miss any degrees of freedom.

Quotient Space of Spherical Rotations

Exploit classic *quotient property* of the topological spaces of the orthogonal groups

$$SO(N)/SO(N-1) = S^{N-1}$$

where S^K is a K-dimensional topological sphere.

$\Rightarrow$ Fill out $N(N-1)/2$ parameters of $SO(N)$, (N-dimensional orthogonal rotations), as a *nested family of points on spheres*.

(See paper for details.)

Interpolating on Sphere

Classic building block of uniform-angular-velocity interpolation is a constant angular velocity spherical interpolation, the "Slerp" between two directions, $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$:

$$\begin{aligned} \hat{\mathbf{n}}_{12}(t) &= \text{Slerp}(\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2, t) \\ &= \hat{\mathbf{n}}_1 \frac{\sin((1-t)\theta)}{\sin(\theta)} + \hat{\mathbf{n}}_2 \frac{\sin(t\theta)}{\sin(\theta)} \end{aligned}$$

where $\cos\theta = \hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2$.

(This formula is simply the result of applying a Gram-Schmidt decomposition while enforcing unit norm in *any dimension*.)

Quaternion form in 3D

Quaternions allow geodesic approach to paths more complex than single Slerp, which is the same in either 3×3 or quaternion form.

If $\hat{\mathbf{n}}$ is a unit 3-vector, define quaternion

$$q_0 = \cos(\theta/2), \quad \vec{\mathbf{q}} = \hat{\mathbf{n}} \sin(\theta/2)$$

This is automatically a point on S^3 due to the constraint $(q_0)^2 + (q_1)^2 + (q_2)^2 + (q_3)^2 = 1$. Then each point q on S^3 corresponds to an $SO(3)$ rotation matrix R_3:

$$\begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{bmatrix}$$

Quaternion form in 4D

4D rotation group $SO(4)$ and the spin group Spin(4) that is its double covering may be computed by extending quaternion multiplication to act not just on 3-vectors ("pure" quaternions) $v = (0, \vec{\mathbf{V}})$, but on full 4-vector quaternions v^μ in the following way:

$$\sum_{\nu=0}^{3} R^\mu{}_\nu v^\nu = q \cdot v^\mu \cdot p^{-1} \, .$$

Double quaternion parameterization of 4D rotations takes the form of the following matrix R_4:

$$\begin{bmatrix} q_0p_0 + q_1p_1 + q_2p_2 + q_3p_3 & q_1p_0 - q_0p_1 - q_3p_2 + q_2p_3 \\ -q_1p_0 + q_0p_1 - q_3p_2 + q_2p_3 & q_0p_0 + q_1p_1 - q_2p_2 - q_3p_3 \\ -q_2p_0 + q_0p_2 - q_1p_3 + q_3p_1 & q_1p_2 + q_2p_1 + q_0p_3 + q_3p_0 \\ -q_3p_0 + q_0p_3 - q_2p_1 + q_1p_2 & q_1p_3 + q_3p_1 - q_0p_2 - q_2p_0 \end{bmatrix}$$

$$\begin{bmatrix} q_2p_0 - q_0p_2 - q_1p_3 + q_3p_1 & q_3p_0 - q_0p_3 - q_2p_1 + q_1p_2 \\ q_1p_2 + p_1q_2 - p_0q_3 - q_0p_3 & q_1p_3 + p_1q_3 + p_0q_2 + q_0p_2 \\ q_0p_0 + q_2p_2 - q_1p_1 - q_3p_3 & q_2p_3 + q_3p_2 - q_0p_1 - q_1p_0 \\ q_2p_3 + q_3p_2 + q_1p_0 + p_0q_1 & q_0p_0 + q_3p_3 - q_1p_1 - q_2p_2 \end{bmatrix}$$

Analogs of the $N = 3$ and $N = 4$ approaches for general N involve computing Spin(N) geodesics and thus are quite complex.

Arcball and Virtual Sphere Controls

- **4D roll.** Elegant user interface for all 6 DOF using 3D position of wand, flying mouse, etc.
- **4D virtual sphere.** 4D roll at center, 3D roll at boundaries of solid sphere.
- **3D roll/sphere pairs.** Using the decomposition of a 4D rotation into two 3D rotations, one can perform a pair of 3D rotations in turn.
- **3D Arcball pairs.** Shoemake 3D arcball is similarly generalizable by picking *two pairs* of points instead of one pair; compose two ordinary rotations into one.
- $N > 4$ **Problems.** Extending to $N > 4$ will require finding elegant ways of computing geodesics in the spin groups corresponding to each orthogonal group. Clifford algebras appear to handle the group properties, but explicit geodesic-like splines in these variables are unknown at present.

Part II(A): Quaternion Frames

Andrew J. Hanson

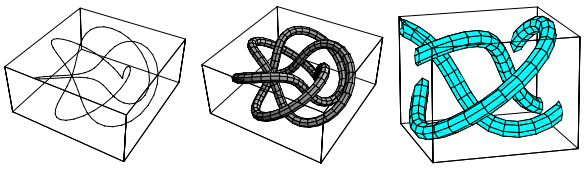
Computer Science Department

Indiana University

Outline

- **Motivation**
- **2D Curves:** A simple “frame”-work
- **3D Curves:** Frenet and Parallel Transport Frames
- **Quaternion Frames** for Curves and Surfaces

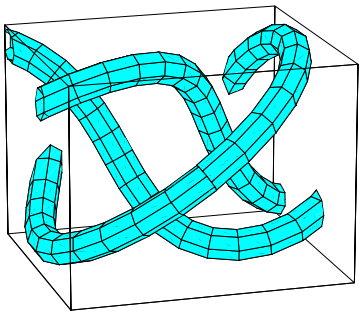
Motivating Problems: Curves



The (3,5) torus knot.

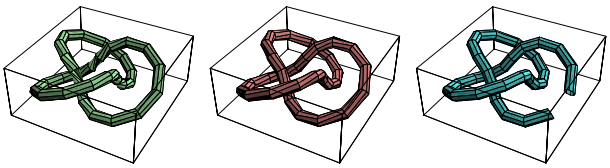
- Line drawing ≈ useless.
- Tubing based on parallel transport, **not periodic.**
- Closeup of the non-periodic mismatch.

Motivating Problems: Curves



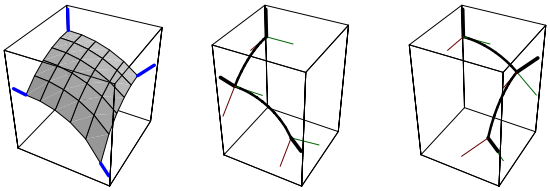
Closeup of the non-periodic mismatch.
Can’t apply texture.

Motivating Problems: Curves



Tubings based on Frenet, Geodesic Reference, and Parallel Transport frames.

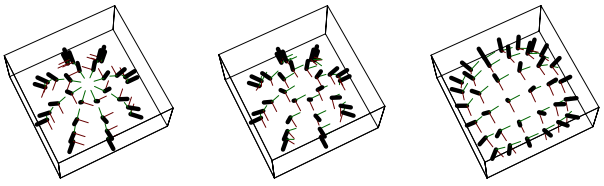
Motivating Problems: Surfaces



A smooth 3D surface patch: two ways to get bottom frame.

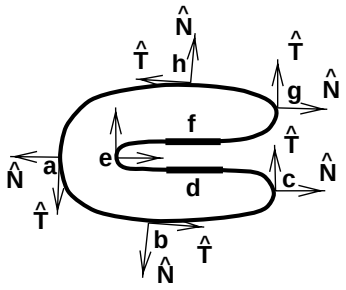
No unique orthonormal frame is derivable from the parameterization.

Motivating Problems: Surfaces



Spherical surface patch.
Frames from Polar, Geodesic Reference, and Projective coordinates.

Frames in 2D



Tangent and *normal* of 2D curve $\mathbf{x}(t)$.

$$\begin{aligned} \mathbf{T}(t) &= d\mathbf{x}(t)/dt = x'\hat{\mathbf{x}} + y'\hat{\mathbf{y}} \\ \mathbf{N}(t) &= y'\hat{\mathbf{x}} - x'\hat{\mathbf{y}} \ . \end{aligned}$$

Unit length vector notation: $\hat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|$.

The column vectors $\hat{\mathbf{N}}$ and $\hat{\mathbf{T}}$ then represent a moving orthonormal coordinate frame.

Frame Evolution in 2D

Demonstrate concepts of 3D frames in simpler 2D context: show how the 2D coordinate frames evolve:

$$\left[\begin{array}{cc}\hat{\mathbf{N}} & \hat{\mathbf{T}}\end{array}\right] = \left[\begin{array}{cc}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{array}\right].$$

Differentiate to find frame equations:

$$\begin{aligned}\hat{\mathbf{N}}'(t) &= +\kappa\hat{\mathbf{T}} \\ \hat{\mathbf{T}}'(t) &= -\kappa\hat{\mathbf{N}}\,,\end{aligned}$$

where $\kappa(t) = d\theta/dt$ is the **curvature**.

This is 2D analog of 3D *Parallel Transport Frame*.

2D "Quaternion" Frames

We are not done yet: we can express this $SO(2)$ frame equation in terms of the *spin group Spin*(2). Guess a double-valued quadratic form in (a, b):

$$\left[\hat{\mathbf{N}} \quad \hat{\mathbf{T}}\right] = \left[\begin{array}{cc}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{array}\right] = \left[\begin{array}{cc}a^2-b^2 & -2ab \\ 2ab & a^2-b^2\end{array}\right],$$

where imposing the constraint $a^2+b^2 = 1$ guarantees orthonormality of the frame.

Solution: *double-angle formulas*:

$$a = \cos(\theta/2),\ b = \sin(\theta/2)$$

2D Quaternions . . .

The matrix equation

$$\left[\begin{array}{c}a' \\ b'\end{array}\right] = \frac{1}{2}\left[\begin{array}{cc}0 & -\kappa \\ +\kappa & 0\end{array}\right]\cdot\left[\begin{array}{c}a \\ b\end{array}\right]$$

in the two variables with one constraint contains *both* the frame equations $\hat{\mathbf{T}}' = -\kappa\hat{\mathbf{N}}$ and $\hat{\mathbf{N}}' = +\kappa\hat{\mathbf{T}}$.

This is square root of frame equations.

But if we let $(a + ib) = \exp(i\,\theta/2)$ we see that rotation is complex multiplication and quaternion frames in 2D are just complex numbers, with

evolution eqns = derivative of $\exp(i\,\theta/2)$!

3D Curves: Frenet and PT Frames

Classic Moving Frame:

$$\begin{bmatrix}\mathbf{T}'(t) \\ \mathbf{N}'(t) \\ \mathbf{B}'(t)\end{bmatrix} = \begin{bmatrix}0 & k_1(t) & k_2(t) \\ -k_1(t) & 0 & \sigma(t) \\ -k_2(t) & -\sigma(t) & 0\end{bmatrix}\begin{bmatrix}\mathbf{T}(t) \\ \mathbf{N}(t) \\ \mathbf{B}(t)\end{bmatrix}.$$

Serret-Frenet frame: $k_2 = 0$, $k_1 = \kappa(t)$ is the curvature, and $\sigma(t) = \tau(t)$ is the classical torsion. LOCAL.

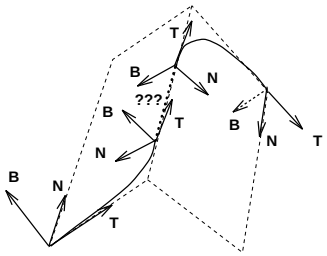
Parallel Transport frame (Bishop): $\sigma = 0$ to get minimal turning. NON-LOCAL = an INTEGRAL.

3D curve frames, contd

Frenet frame is *locally* defined, e.g., by

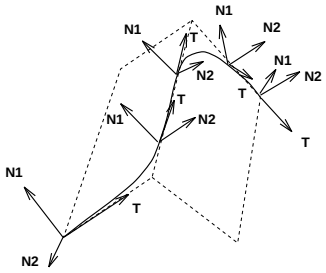
$$\mathbf{B}(t) = \frac{\mathbf{x}'(t) \times \mathbf{x}''(t)}{\|\mathbf{x}'(t) \times \mathbf{x}''(t)\|}$$

but has problems on the "roof."



3D curve frames, contd

Bishop's Parallel Transport frame is *integrated over whole curve*, non-local, but no problems on "roof:"



Quaternion Frames

We can now repeat our trick of taking the square root of the 2D frame using *quaternions* to generalize the single 2D rotation angle.

Summary of Quaternion Frame properties:

- **Unit four-vector.** Take $q = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{q})$ to obey constraint $q \cdot q = 1$.
- **Multiplication rule.** Let $q * p$ be the quaternion product of two quaternions q and p, where

$$\begin{bmatrix} [q*p]_0 \\ [q*p]_1 \\ [q*p]_2 \\ [q*p]_3 \end{bmatrix} = \begin{bmatrix} q_0p_0 - q_1p_1 - q_2p_2 - q_3p_3 \\ q_0p_1 + q_1p_0 + q_2p_3 - q_3p_2 \\ q_0p_2 + q_2p_0 + q_3p_1 - q_1p_3 \\ q_0p_3 + q_3p_0 + q_1p_2 - q_2p_1 \end{bmatrix}$$

  This $=$ multiplication rule in the group $\mathrm{SU}(2)$, the double covering of $\mathrm{SO}(3)$ rotation group.

Quaternion Frames . . .

Quaternion Frame properties, contd:

- **Rotation Correspondence.** The unit quaternions q and $-q$ correspond to a single 3D rotation:

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

- **Rotation Correspondence.** Let $q = (\cos\frac{\theta}{2}, \hat{\mathbf{n}}\sin\frac{\theta}{2})$, with $\hat{\mathbf{n}}$ a unit 3-vector, $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$. Then $R(\theta, \hat{\mathbf{n}})$ is usual 3D rotation by θ in the plane perpendicular to $\hat{\mathbf{n}}$.
- **Inversion.** Any 3×3 matrix R can be inverted for q *up to a sign.* Carefully treat singularities! Can choose sign, e.g., by local consistency, to get continuous frames.

Quaternion Frames . . .

Now find *square root* of 3D frame eqns:

Tait (1890) derived a quaternion equation that makes **all 9 3D frame equations reduce to**:

$$\begin{bmatrix} q_0' \\ q_1' \\ q_2' \\ q_3' \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\sigma & -k_2 & -k_1 \\ \sigma & 0 & k_1 & k_2 \\ k_2 & -k_1 & 0 & \sigma \\ k_1 & -k_2 & -\sigma & 0 \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

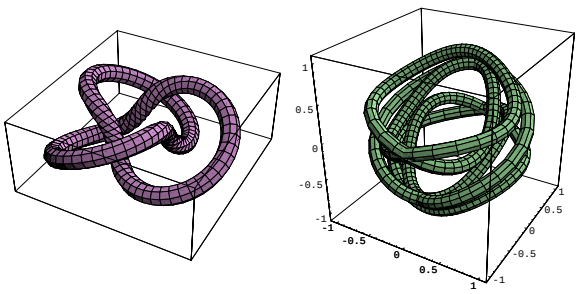
Quaternion Frames . . .

Properties of Tait's quaternion frame equations:

- Antisymmetry $\Rightarrow q(t) \cdot q'(t) = 0$.
- *Nine equations and six constraints* become *four equations and one constraint*, keeping quaternion on the 3-sphere. $\Rightarrow$ **Good for computer implementation.**
- Analogous treatment applies to Weingarten equations, allowing a direct quaternion treatment of the classical differential geometry of *surfaces* as well.

Quaternion Frames, . . .

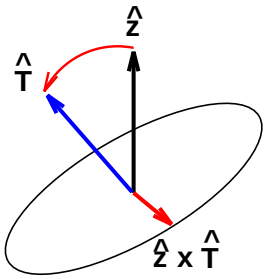
Example: torus knot and its (twice around) quaternion Frenet frame:



see: Hanson and Ma, "Quaternion Frame Approach to Streamline Visualization," *IEEE Trans. on Visualiz. and Comp. Graphics*, **1**, No. 2, pp. 164–174 (June, 1995).

Geometric Construction of Space of Frames:

- $R(\theta, \hat{\mathbf{T}})$ leaves $\hat{\mathbf{T}}$ invariant, but doesn't have $\hat{\mathbf{T}}$ as Last Column.
- Use Geodesic Reference to construct one instance of such a frame: $R(\hat{\mathbf{z}} \cdot \hat{\mathbf{T}}, \hat{\mathbf{z}} \times \hat{\mathbf{T}})$.
- $q(\theta, \hat{\mathbf{T}}) * q(\hat{\mathbf{z}} \cdot \hat{\mathbf{T}}, \hat{\mathbf{z}} \times \hat{\mathbf{T}})$ generates the correct family of quaternion curves

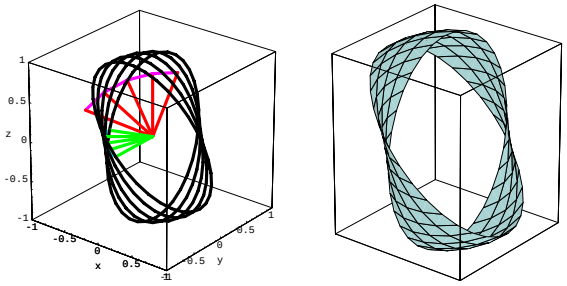


Invariant Quaternion Frames . . .

Invariant frame for trefoil knot:

- Left: Red fan = tangents; Magenta arc = tangent map; Green vectors = geodesic reference starting points for invariant spaces. Right: Short segment of invariant space.

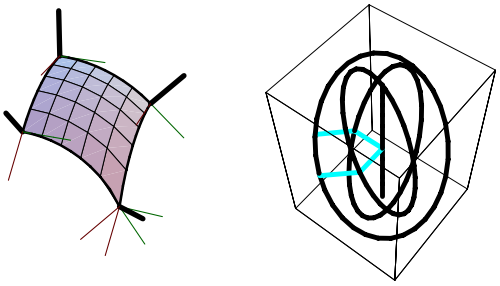
- Full Space.



3-Manifold of Frames for a Patch

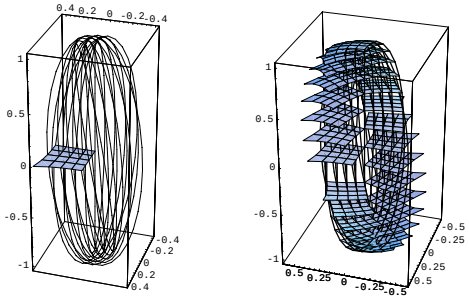
For surfaces, we simply replace a curve's tangent by a surface's normal.

Basic patch with the available rings of frames for corners:



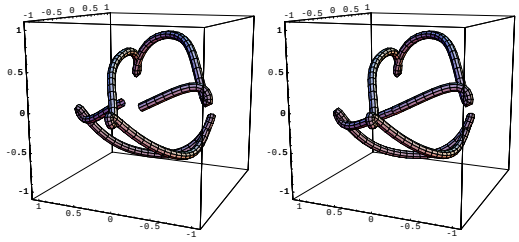
3-manifold of frames for a patch . . .

Each point on patch generates a ring in quaternion map:



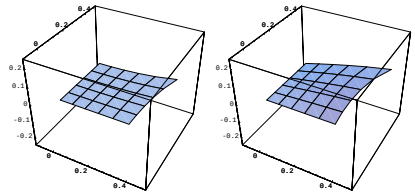
Minimizing Quaternion Length Solves Periodic Tube

Quaternion space optimization of the non-periodic parallel transport frame of the (3,5) torus knot.



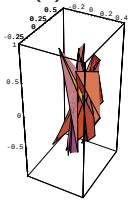
Likewise for Optimal Quaternion Frame on Patch

Quaternion frames for (a) Geodesic Ref. (b) One edge Parallel Transport. (c) Random. (d) Minimal area result.

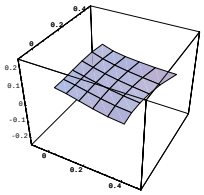


(a)

(b)



(c)



(d)

Summary

- **Quaternions can represent frames.**
- **Curve frames $\Rightarrow$ quaternion curves.**
- **Surface patch frames $\Rightarrow$ quaternion surface patches.**
- **Minimizing quaternion length or area finds parallel transport "minimal turning" set of frames.**

Part II(B): 4D Rendering and 3D Scalar Fields

Andrew J. Hanson

Computer Science Department

Indiana University

Outline

- **Shading volumetric objects embedded in 4D.**
- **Extensions:** Rendering thin 4D objects by thickening analogy.
- **Video: "FourSight"**
- **4D Terrain.** Using 4D rotation and 4D lighting to extend standard 3D volume rendering.
- **Video: "4D Views of 3D Scalar Fields"**

4D Shading

Need the following pieces:

- **Tesselation.** Describe volumes using tetrahedra with 4D verts, or *carefully* subdivide a cubic lattice.
- **Normals.** Compute 4D normals to tetrahedra (empty-right-column determinant).
- **Light and Camera.** Dot product of light with 4D versions of usual vectors give Gouraud and Phong models. (Banks: Modulate power in Phong?)
- **Projection.** Film $\approx$ volumetric retina of 4D cyclops.
- **Volume Render.** Splat, Shirley-Tuchman, texture panels; motion very important. Stereo impossible without volumetric texture.

Can You Believe 4D Shading?

Consider the *Visualization Principle* for testing the potential of a visualization:

- Volume rendered projections of 4D lit hypersurfaces to 3D contain information similar to 2D magazine photo.
- Humans and smart machine vision programs can reconstruct 3D info from magazine using 2D *shape from shading*.
- Therefore it is *possible* to reconstruct significant 4D information from the proposed rendering methods: use 3D shape from shading.

Extensions to 4D Rendering

A 4D object need not be volumetric to be rendered!

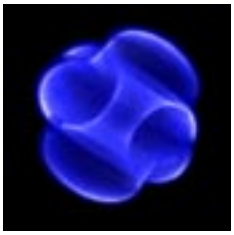
- A wire in 3D can be made visible by attaching a *shiny tube*.
- **Imitate in 4D:** Surfaces $\Rightarrow$ handled by "tubing" with a shiny circle attached to each point, forcing it to be volumetric and thus renderable.
- Curves have small spheres attached, and even *points* can be made structurally visible in 4D by making them tiny 3-spheres!

Efficient Approximations to 4D Surface Rendering

A 4D surface can be rendered directly, without attaching a tube to create a thickening.

- In 3D, the Kajiya-Kay "bear hair" algorithm suggests how to replace an expensive *shiny tube* by a physically plausible reflectance texture.
- **Imitate in 4D:** Calculate the light that would be reflected from a surface if you attached shiny circles and shrunk them to a point.
- Can derive various heuristic lighting models for both Phong and Gouraud (Banks, Cross, Hanson).
- **Render surface** projected to 3D from 4D directly by applying (possibly transparent) texture map.
- *Avoids volume rendering altogether,* yielding large speedup.

Examples of 4D Rendering




4D "knotted surfaces," thickened to make a volume, then lit, projected to 3D "film," and volume rendered.


The same 4D "knotted surfaces," but "bear hair" approximation producing a transparent surface texture *simulating* volume rendering 1000 times as fast.

3D Scalar Field Data

A 3D scalar field is a collection of single numbers $w = f(x, y, z)$ assigned throughout a spatial volume. Conventionally, some of the following methods are used to display this data:

- **Direct volume rendering:** render pixels directly from the screen, e.g., by splatting, or by forward or backward ray-tracing.
- **Pseudocolor scalar field values.** — color code the interior using the data values.
- **Slicing:** — Look at textured slices, preferably moving, at various angles to the data.
- **Isosurfaces, Marching Cubes:** — Look at surfaces that are computed by connecting the interior points that seem to have have the same data value, stitch together the total.
- **Break into tiny cubes.** — to help see inside volume, leave a little space instead of transparency

4D Extensions to 3D Scalar Field Methods

3D scalar fields are to 4D display methods as 2D elevation maps are to 3D terrain display methods.

Thus the following new ideas arise:

- **Rotate in 4D.** — to "look obliquely" at the data values, as though seeing mountains from an unfamiliar valley.
- *New Cues:* Ridges, occlusion edges, depth from occlusion. Isosurface markings and colorization may be added.
- **Illuminate data in 4D.** — use methods just learned for computing normals and lighting effects; rotate light and also object to get motion parallax.
- *New Cues:* Shading gives constraints on direction of normal of the terrain surface. Shininess and lighting effects make it easier to characterize the surface, materials, etc. Lighting shows general directions of lights; lighting on objects produces strong depth using highly directional shininess cues.

Remarks: Multiple cues

3D terrain models are often displayed in oblique views with multiple cues such as contours:

- **Multiple cues.** Our 4D terrain models could also be enhanced by multiple cues such as color, slices, and isosurfaces.
- **Cubic Lattice Grids.** Just as secondary depth cues such as lattices or checkerboard patterns arise naturally on a 2D terrain, we could apply a cubic checkerboard-like texture to the terrain.
- **What else can we try?** Quaternion forms of rotations and the display of 3D scalar fields just happened to appear as useful application areas of ND geometry; we would love to hear about other such domains of research!

◊ II.6

Geometry for N-Dimensional Graphics

Andrew J. Hanson
Computer Science Department
Indiana University
Bloomington, IN 47405
hanson@cs.indiana.edu

◊ Introduction ◊

Textbook graphics treatments commonly use special notations for the geometry of 2 and 3 dimensions that are not obviously generalizable to higher dimensions. Here we collect a family of geometric formulas frequently used in graphics that are easily extendible to N dimensions as well as being helpful alternatives to standard 2D and 3D notations.

What use are such formulas? In mathematical visualization, which commonly must deal with higher dimensions — 4 real dimensions, 2 complex dimensions, etc. — the utility is self-evident (see, e.g., (Banchoff 1990, Francis 1987, Hanson and Heng 1992b, Phillips et al. 1993)). The visualization of statistical data also frequently utilizes techniques of N-dimensional display (see, e.g., (Noll 1967, Feiner and Beshers 1990a, Feiner and Beshers 1990b, Brun et al. 1989, Hanson and Heng 1992a)). We hope that publicizing some of the basic techniques will encourage further exploitation of N-dimensional graphics in scientific visualization problems.

We classify the formulas we present into the following categories: basic notation and the N-simplex; rotation formulas; imaging in N-dimensions; N-dimensional hyperplanes and volumes; N-dimensional cross-products and normals; clipping formulas; the point-hyperplane distance; barycentric coordinates and parametric hyperplanes; N-dimensional ray-tracing methods. An appendix collects a set of obscure Levi-Civita symbol techniques for computing with determinants. For additional details and insights, we refer the reader to classic sources such as (Sommerville 1958, Coxeter 1991, Hocking and Young 1961) and (Banchoff and Werner 1983, Efimov and Rozendorn 1975).

◊ Definitions — What is a Simplex, Anyway? ◊

In a nutshell, an N-simplex is a set of $(N+1)$ points that together specify the simplest non-vanishing N-dimensional volume element (e.g., two points delimit a line segment in 1D, 3 points a triangle in 2D, 4 points a tetrahedron in 3D, etc.). From a mathematical point of view,

ISBN 0-12-336156-7

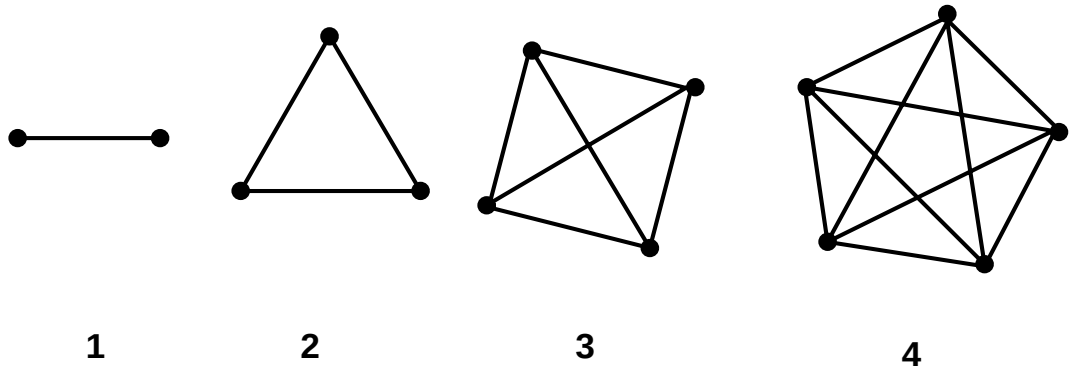


Figure 1. 2D projections of simplexes with dimension 1–4. An N-simplex is defined by $(N+1)$ linearly independent points and generalizes the concept of a line segment or a triangular surface patch.

there are lots of different N-dimensional spaces: here we will restrict ourselves to ordinary flat, real Euclidean spaces of N dimensions with global orthogonal coordinates that we can write as

$$\vec{x} = (x, y, z, \ldots, w)$$

or more pedantically as

$$\vec{x} = (x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(N)}) \; .$$

We will use the first, less cumbersome, notation whenever it seems clearer.

Our first type of object in N-dimensions, the 0-dimensional *point* $\vec{x}$, may be thought of as a vector from the origin to the designated set of coordinate values. The next type of object is the 1-dimensional *line*, which is determined by giving two points $(\vec{x}_0, \vec{x}_1)$; the line segment from $\vec{x}_0$ to $\vec{x}_1$ is called a 1-*simplex*. If we now take three noncollinear points $(\vec{x}_0, \vec{x}_1, \vec{x}_2)$, these uniquely specify a *plane*; the triangular area delineated by these points is a 2-*simplex*. A 3-simplex is a solid tetrahedron formed by a set of four noncoplanar points, and so on. In figure 1, we show schematic diagrams of the first few simplexes projected to 2D.

Starting with the $(N+1)$ points $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$ defining a simplex, one then connects all possible pairs of points to form edges, all possible triples to form faces, and so on, resulting in the structure of component "parts" given in table 1. The next higher object uses its predecessor as a building block: a triangular face is built from three edges, a tetrahedron is built from four triangular faces, a 4-simplex is built from 5 tetrahedra.

The general idea should now be clear: $(N+1)$ linearly independent points define a *hyperplane* of dimension N and specify the boundaries of an N-dimensional coordinate patch comprising an N-*simplex* (Hocking and Young 1961). Just as the surfaces modeling a 3D object may be broken up (or *tessellated*) into triangular patches, N-dimensional objects may be tessellated into $(N-1)$-dimensional simplexes that define their geometry.

Table 1. Numbers of component structures making up an N-simplex. For example, in 2D, the basic simplex is the triangle with 3 points, 3 edges, and one 2D face.

	Dimension of Space					
Type of Simplex	$N=1$	$N=2$	$N=3$	$N=4$	...	N
Points (0D)	2	3	4	5	...	$\binom{N+1}{1} = N+1$
Edges (1D simplex)	1	3	6	10	...	$\binom{N+1}{2}$
Faces (2D simplex)	0	1	4	10	...	$\binom{N+1}{3}$
Volumes (3D simplex)		0	1	5	...	$\binom{N+1}{4}$
⋮	⋮	⋮	⋮	⋮	⋱	⋮
$(N-2)$D simplex					...	$\binom{N+1}{N-1}$
$(N-1)$D simplex					...	$\binom{N+1}{N} = N+1$
ND simplex				1	...	$\binom{N+1}{N+1} = 1$

♢ Rotations ♢

In N Euclidean dimensions, there are $\binom{N}{2} = N(N-1)/2$ degrees of rotational freedom corresponding to the free parameters of the group $SO(N)$. In 2D, that means we only have one rotational degree of freedom given by the angle used to mix the x and y coordinates. In 3D, there are 3 parameters, which can be thought of as corresponding either to three Euler angles or to the three independent quaternion coordinates that remain when we represent rotations in terms of unit quaternions. In 4D, there are 6 degrees of freedom, and the familiar 3D picture of "rotating about an axis" is no longer valid; each rotation leaves an entire plane fixed, not just one axis.

General rotations in N dimensions may be viewed as a sequence of elementary rotations. Each elementary rotation acts in the plane of a particular pair, say (i,j), of coordinates, leaving an $(N-2)$-dimensional subspace unchanged; we may write any such rotation in the form

$$\begin{aligned} x'^{(i)} &= x^{(i)}\cos\theta \pm x^{(j)}\sin\theta \\ x'^{(j)} &= \mp x^{(i)}\sin\theta + x^{(j)}\cos\theta \\ x'^{(k)} &= x^{(k)} \quad (k \neq i,j)\ . \end{aligned}$$

It is important to remember that *order matters* when doing a sequence of nested rotations; for example, two sequences of small 3D rotations, one consisting of a $(2,3)$-plane rotation followed

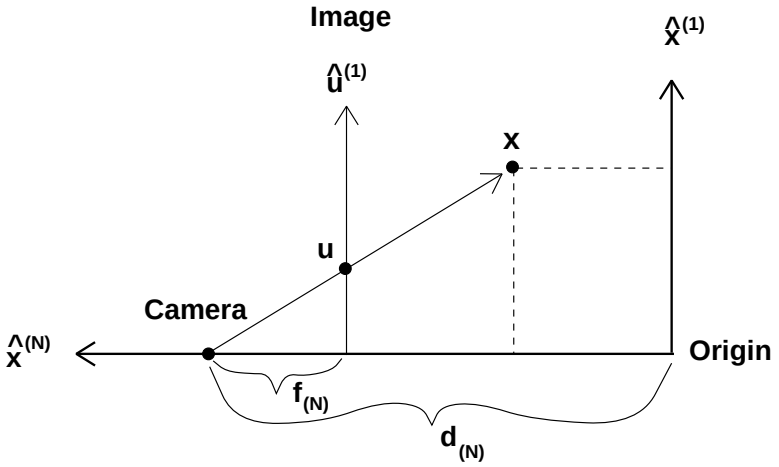


Figure 2. Schematic view of the projection process for an N-dimensional pinhole camera.

by a $(3,1)$-plane rotation, and the other with the order reversed, will differ by a rotation in the $(1,2)$-plane. (See any standard reference such as (Edmonds 1957).)

We then have a number of options for controlling rotations in N-dimensional Euclidean space. Among these are the following:

- (i,j)**-space pairs.** A brute-force choice would be just to pick a sequence of (i,j) planes in which to rotate using a series of matrix multiplications.
- (i,j,k)**-space triples.** A more interesting choice for an interactive system is to provide the user with a family of (i,j,k) triples having a 2D controller like a mouse coupled to two of the degrees of freedom, and having the 3rd degree of freedom accessible in some other way — with a different button, from context using the "virtual sphere" algorithm of (Chen et al. 1988), or implicitly using a context-free method like the "rolling-ball" algorithm (Hanson 1992). The simplest example is $(1,2,3)$ in 3D, with the mouse coupled to rotations about the $\hat{\mathbf{x}}$-axis $(2,3)$ and the $\hat{\mathbf{y}}$-axis $(3,1)$, giving $\hat{\mathbf{z}}$-axis $(1,2)$ rotations as a side-effect. In 4D, one would have four copies of such a controller, $(1,2,3)$, $(2,3,4)$, $(3,1,4)$, and $(1,2,4)$, or two copies exploiting the decomposition of $SO(4)$ infinitesimal rotations into two independent copies of ordinary 3D rotations. In N dimensions, $\left(\begin{array}{c} N \\ 3 \end{array}\right)$ sets of these controllers (far too many when N is large!) could in principle be used.

♢ N-dimensional Imaging ♢

The general concept of an "image" is a projection of a point $\vec{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(N)})$ from dimension N to a point $\vec{u}$ of dimension $(N-1)$ along a line. That is, the image of a 2D world

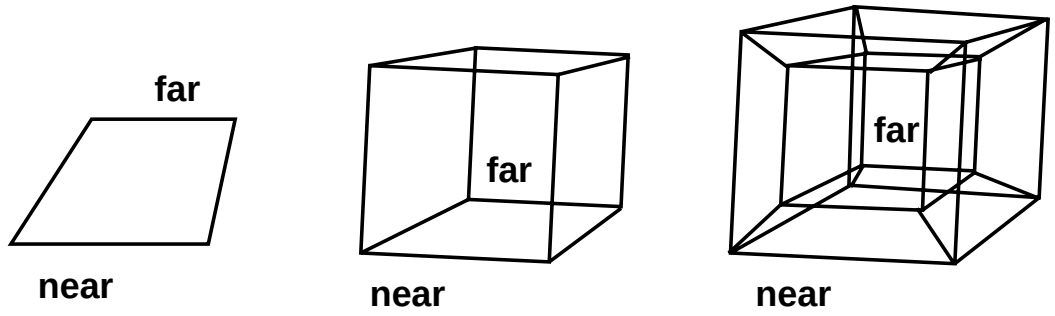


Figure 3. Qualitative results of perspective projection of a wire-frame square, a cube, and a hypercube in 2D, 3D, and 4D, respectively.

is a projection to 1D film, 3D worlds project to 2D film, 4D worlds project to 3D film, and so on. Since we can rotate our coordinate system as we please, we lose no generality if we assume this projection is along the N-th coordinate axis. An orthographic or parallel projection results if we simply throw out the N-th coordinate $x^{(N)}$ of each point. A pinhole camera perspective projection (see figure 2) results when, in addition, we scale the first $(N-1)$ coordinates by dividing by $(d_N - x^{(N)})/f_N$, where d_N is the distance along the positive N-th axis to the camera focal point and f_N is the focal length. One may need to project this first image to successively lower dimensions to make it displayable on a 2D graphics screen; thus a hierarchy of up to $(N-2)$ parameter sets $\{(f_N, d_N), \ldots, (f_3, d_3)\}$ may be introduced if desired.

In the familiar 3D case, we replace a vertex (x, y, z) of an object by the 2D coordinates $(xf/(d-z), yf/(d-z))$, so that more distant objects (in the negative z direction) are shrunk in the 2D image. In 4D, entire solid objects are shrunk, thus giving rise to the familiar wire-frame hypercube shown in figure 3 that has the more distant cubic hyperfaces actually lying *inside* the projection of the nearest cube.

As we will see a bit later when we discuss normals and cross-products, the usual shading approaches allow only $(N-1)$-manifolds to interact uniquely with a light ray. That is, the generalization of a viewable "object" to N dimensions is a manifold of dimension $(N-1)$ that bounds an N-dimensional volume; only this boundary is visible in the projected image if the object is opaque. For example, curves in 2D reflect light toward the focal point to form images on a "film line," surface patches in 3D form area images on a 2D film plane, volume patches in 4D form volume images in the 3D film volume, etc. The image of this $(N-1)$-dimensional patch may be ray traced or scan converted. Objects are typically represented as tessellations which consist of a collection of $(N-1)$-dimensional simplexes; for example, triangular surface patches form models of the visible parts of 3D objects, while tetrahedral volumes form models of the visible parts of 4D objects. (An interesting side issue is how to display meaningful illuminated images of lower dimensional manifolds — lines in 3D, surfaces and lines in 4D, etc.; see (Hanson and Heng 1992b) for further discussion.)

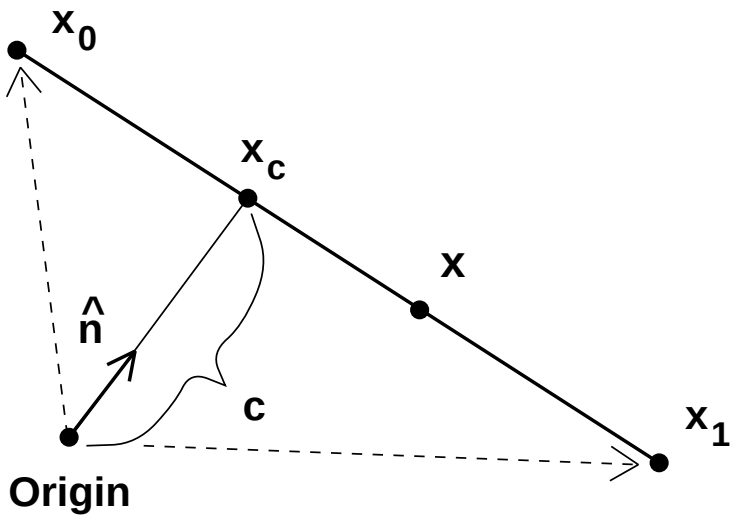


Figure 4. The line from $\vec{x}_0$ to $\vec{x}_1$ whose points obey the equation $\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = 0$. The constant c is just $\hat{\mathbf{n}} \cdot \vec{x}_0$.

♢ Hyperplanes and Volume Formulas ♢

Implicit Equation of a Hyperplane. In 2D, a special role is played by the single linear equation defining a line; in 3D, the analogous single linear equation defines a plane. In N-dimensions, the following implicit linear equation describes a set of points belonging to an $(N-1)$-dimensional hyperplane:

$$\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = 0\,. \tag{1}$$

Here $\vec{x}_0$ is any point on the hyperplane and conventionally $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$. The geometric interpretation of this equation in 2D is the 1D line shown in figure 4. In general, $\hat{\mathbf{n}}$ is a normalized unit vector that is perpendicular to the hyperplane, and $\hat{\mathbf{n}} \cdot \vec{x}_0 = c$ is simply the (signed) distance from the origin to the hyperplane. The point $\vec{x}_c = c\hat{\mathbf{n}}$ is the point on the hyperplane closest to the origin; the point closest to some other point $\vec{P}$ is $\vec{x}_c = \vec{P} + \hat{\mathbf{n}}\{\hat{\mathbf{n}} \cdot (\vec{x}_0 - \vec{P})\}$.

Simplex Volumes and Subvolumes. The volume (by which we always mean the N-dimensional hypervolume) of an N-simplex is determined in a natural way by a determinant of its $(N+1)$ defining points (Sommerville 1958):

$$V_N = \frac{1}{N!} \det \begin{bmatrix} x_1 & x_2 & \cdots & x_N & x_0 \\ y_1 & y_2 & \cdots & y_N & y_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_1 & w_2 & \cdots & w_N & w_0 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} . \tag{2}$$

The bottom row of 1's in eq. (2) corresponds to the familiar homogeneous coordinate used with 4×4 projection matrices in 3D graphics. We will attempt to convince the reader in a moment that disastrous sign inconsistencies result unless the global origin $\vec{x}_0$ of the N-simplex's coordinate system is in the last column as shown.

The expression for the volume in eq. (2) is *signed*, which means that it implicitly defines the N-dimensional generalization of the *Right-Hand Rule* typically adopted to determine triangle orientation in 3D geometry. For example, we observe that if $\vec{x}_0 = (0, 0, \ldots, 0)$ is the origin and we choose $\vec{x}_1 = (1, 0, \ldots, 0)$, $\vec{x}_2 = (0, 1, 0, \ldots, 0)$, and so on, the value of the determinant is $+1$. If we had put $\vec{x}_0$ in the first row in eq. (2), the sign would alternate from dimension to dimension! We will exploit this signed determinant shortly to define N-dimensional normal vectors, and again later to formulate N-dimensional clipping.

First, we use the standard column-subtraction identity for determinants to reduce the dimension of the determinant in eq. (2) by one, expressing it in a form that is manifestly *translation-invariant*:

$$\begin{aligned} V_N &= \frac{1}{N!} \det \begin{bmatrix} (x_1 - x_0) & (x_2 - x_0) & \cdots & (x_N - x_0) & x_0 \\ (y_1 - y_0) & (y_2 - y_0) & \cdots & (y_N - y_0) & y_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (w_1 - w_0) & (w_2 - w_0) & \cdots & (w_N - w_0) & w_0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \\ &= \frac{1}{N!} \det \begin{bmatrix} (x_1 - x_0) & (x_2 - x_0) & \cdots & (x_N - x_0) \\ (y_1 - y_0) & (y_2 - y_0) & \cdots & (y_N - y_0) \\ \vdots & \vdots & \ddots & \vdots \\ (w_1 - w_0) & (w_2 - w_0) & \cdots & (w_N - w_0) \end{bmatrix} . \end{aligned} \tag{3}$$

These formulas for V_N can be intuitively understood as generalizations of the familiar 3D triple scalar product,

$$[(\vec{x}_1 - \vec{x}_0) \times (\vec{x}_2 - \vec{x}_0)] \cdot (\vec{x}_3 - \vec{x}_0) ,$$

which gives the volume of the parallelepiped with sides $((\vec{x}_1 - \vec{x}_0), (\vec{x}_2 - \vec{x}_0), (\vec{x}_3 - \vec{x}_0))$. The corresponding tetrahedron with vertices at the points $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3)$ has one-sixth the volume of the parallelepiped. The analogous observation in N dimensions is that the factor of $1/N!$ in eq. (3) is the proportionality factor between the volume of the N-simplex and the volume of the parallelepiped whose edges are given by the matrix columns.

Invariance. The volume determinant is invariant under rotations. To see this explicitly, let $|X|$ be the matrix in eq. (3) and let $|R|$ be any orthonormal rotation matrix (i.e., one whose columns are of unit length and are mutually perpendicular, with unit determinant); then, letting $|X'| = |R| \cdot |X|$, we find

$$\det |X'| = \det(|R| \cdot |X|) = \det |R| \det |X| = \det |X| = N!\, V_N ,$$

since the determinant of a product is the product of the determinants.

A manifestly translation *and* rotation invariant form for the square of the volume element is

$$\begin{aligned}(V_N)^2 &= \left(\frac{1}{N!}\right)^2 \det|X^t \cdot X| \\ &= \left(\frac{1}{N!}\right)^2 \det\begin{bmatrix} v(1,1) & v(1,2) & \cdots & v(1,N) \\ v(2,1) & v(2,2) & \cdots & v(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ v(N,1) & v(N,2) & \cdots & v(N,N) \end{bmatrix}, \end{aligned} \tag{4}$$

where $v(i,j) = (\vec{x}_i - \vec{x}_0) \cdot (\vec{x}_j - \vec{x}_0)$.

This invariant form is not presented as an idle observation; we now exploit it to show how to construct volume forms for *subspaces* of N-dimensional spaces, for which the defining vertices of the desired simplex cannot form square matrices!

The trick here is to note that while V_K, for $K < N$, is not expressible in terms of a square matrix of coordinate differences the way V_N is, we may write V_K as the determinant of a square matrix in one particular coordinate frame, and multiply this matrix by its transpose to get a form like eq. 4, which does not depend on the frame. Since the form is invariant, we can transform back to an arbitrary frame to find the following expression for V_K in terms of its K basis vectors $(\vec{x}_k - \vec{x}_0)$ of dimension N:

$$\begin{aligned}(V_K)^2 &= \left(\frac{1}{K!}\right)^2 \det\begin{bmatrix} \vec{x}_1 - \vec{x}_0 \\ \vec{x}_2 - \vec{x}_0 \\ \vdots \\ \vec{x}_K - \vec{x}_0 \end{bmatrix} \cdot \begin{bmatrix} \vec{x}_1 - \vec{x}_0 & \vec{x}_2 - \vec{x}_0 & \cdots & \vec{x}_K - \vec{x}_0 \end{bmatrix} \\ &= \left(\frac{1}{K!}\right)^2 \det\begin{bmatrix} v(1,1) & v(1,2) & \cdots & v(1,K) \\ v(2,1) & v(2,2) & \cdots & v(2,K) \\ \vdots & \vdots & \ddots & \vdots \\ v(K,1) & v(K,2) & \cdots & v(K,K) \end{bmatrix}. \end{aligned} \tag{5}$$

That is, to compute a volume of dimension K in N dimensions, find the K independent basis vectors spanning the subspace, and form a square $K \times K$ matrix of dot products related to V_K^2 by multiplying the $N \times K$ matrix of column vectors by its transpose on the left. When $K = 1$, we see that we have simply the squared Euclidean distance in N dimensions, $v(1,1) = (\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0)$.

♢ Normals and the Cross-Product ♢

A frequently asked question in N-dimensional geometry concerns how to define a normal vector as a cross-product of edges for use in geometry and shading calculations. To begin with,

you must have an $(N-1)$-manifold (a line in 2D, surface in 3D, volume in 4D) in order to have a well-defined normal *vector*; otherwise, you may have a normal *space* (a plane, a volume, etc.). Suppose you have an ordered set of $(N-1)$ edge vectors $(\vec{x}_k - \vec{x}_0)$ tangent to this $(N-1)$-manifold at a point $\vec{x}_0$; typically these vectors are the edges of one of the $(N-1)$-simplexes in the tessellation. Then the normal $\vec{N}$ at the point is a *generalized cross-product* whose components are cofactors of the last column in the following (notationally abusive!) determinant:

$$\begin{aligned}\vec{N} &= N_x\hat{\mathbf{x}} + N_y\hat{\mathbf{y}} + N_z\hat{\mathbf{z}} + \cdots + N_w\hat{\mathbf{w}} \\ &= \det\begin{bmatrix} (x_1-x_0) & (x_2-x_0) & \cdots & (x_{N-1}-x_0) & \hat{\mathbf{x}} \\ (y_1-y_0) & (y_2-y_0) & \cdots & (y_{N-1}-y_0) & \hat{\mathbf{y}} \\ (z_1-z_0) & (z_2-z_0) & \cdots & (z_{N-1}-z_0) & \hat{\mathbf{z}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (w_1-w_0) & (w_2-w_0) & \cdots & (w_{N-1}-w_0) & \hat{\mathbf{w}} \end{bmatrix}. \end{aligned} \tag{6}$$

As usual, we can normalize using $\|\vec{N}\|$, the square root of the sum of the squares of the cofactors, to form the normalized normal $\hat{\mathbf{n}} = \vec{N}/\|\vec{N}\|$. A quick check shows that if the vectors $(\vec{x}_k - \vec{x}_0)$ are assigned to the first $(N-1)$ coordinate axes in order, this normal vector points in the direction of the positive N-th axis. For example, in 2D, we want the normal to the vector $(x_1 - x_0,\ y_1 - y_0)$ to be $\vec{N} = (-(y_1 - y_0),\ (x_1 - x_0))$ so that a vector purely in the x direction has a normal in the positive y direction; placing the column of unit vectors $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}, \ldots, \hat{\mathbf{w}})$ in the *first* column fails this test. The 3D case can be done either way because an even number of columns are crossed! It is tempting to move the column of unit vectors to the first column instead of the last, but one must resist: the choice given here is the one to use for consistent behavior across different dimensions!

The qualitative interpretation of eq. (6) can now be summarized as follows:

- **2D:** Given two points $(\vec{x}_0, \vec{x}_1)$ determining a line in 2D, the cross-product of a *single vector* is the normal to the line.
- **3D:** Given three points defining a plane in 3D, the cross-product of the two 3D vectors outlining the resulting triangle is the familiar formula $(\vec{x}_1 - \vec{x}_0) \times (\vec{x}_2 - \vec{x}_0)$ for the normal $\vec{N}$ to the plane.
- **4D:** In four dimensions, we use four points to construct the three vectors $(\vec{x}_1 - \vec{x}_0), (\vec{x}_2 - \vec{x}_0), (\vec{x}_3 - \vec{x}_0)$; the cross product of these vectors is a *four-vector* that is perpendicular to each vector and thus is interpretable as the normal to the tetrahedron specified by the original four points.

From this point on, the relationship to standard graphics computations should be evident: If, in N-dimensional space, the $(N-1)$-manifold to be rendered is tessellated into $(N-1)$-simplexes, use eq. (6) to compute the normal of each simplex for flat shading. For interpolated shading, compute the normal at each vertex (e.g., by averaging the normals of all neighboring

simplexes and normalizing or by computing the gradient of an implicit function specifying the vertex). Compute the intensity at a point for which you know the normal by taking the dot product of the appropriate illumination vector with the normal (e.g, by plugging it into the last column of eq. (6)). If appropriate, set the dot product to zero if it is negative (pointing away from the light). Back face culling, to avoid rendering simplexes pointing away from the camera, is accomplished in exactly the same way: plug the camera view vector into the last column of eq. (6) and discard the simplex if the result is negative.

Dot Products of Cross Products. We conclude this section with the remark that sometimes computing the dot product between a normal and a simple vector is not enough; if we need to know the relative orientation of two face normals (e.g., to determine whether a finer tessellation is required), we must compute the dot products of normals. In principle, this can be done by brute force directly from eq. (6). Here we note an alternative formulation that is the N-dimensional generalization of the 3D formula for the decomposition of the dot product of two cross products; in the 3D case, if one normal is given by the cross product $\vec{X} = \vec{A} \times \vec{B}$ and the other by $\vec{Y} = \vec{C} \times \vec{D}$, we can write

$$\vec{X} \cdot \vec{Y} = (\vec{A} \times \vec{B}) \cdot (\vec{C} \times \vec{D}) = (\vec{A} \cdot \vec{C})(\vec{B} \cdot \vec{D}) - (\vec{A} \cdot \vec{D})(\vec{B} \cdot \vec{C}) \; . \tag{7}$$

We note that the degenerate case for the square of a cross product is

$$(\vec{A} \times \vec{B}) \cdot (\vec{A} \times \vec{B}) = (\vec{A} \cdot \vec{A})(\vec{B} \cdot \vec{B}) - (\vec{A} \cdot \vec{B})^2 \; ,$$

which, if θ is the angle between $\vec{A}$ and $\vec{B}$, reduces to the identity $\|\vec{A}\|^2 \|\vec{B}\|^2 \sin^2 \theta = \|\vec{A}\|^2 \|\vec{B}\|^2 - \|\vec{A}\|^2 \|\vec{B}\|^2 \cos^2 \theta$.

The generalization of this expression to N dimensions can be derived from the product of two Levi-Civita symbols (see the Appendix). If $\vec{X}$ and $\vec{Y}$ are two cross products formed from the sets of vectors $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{N-1}$ and $\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_{N-1}$, then

$$\begin{aligned} \vec{X} \cdot \vec{Y} \quad = \quad & \sum_{\text{all indices}} x_1^{(i_1)} x_2^{(i_2)} \ldots x_{N-1}^{(i_{N-1})} y_1^{(j_1)} y_2^{(j_2)} \ldots y_{N-1}^{(j_{N-1})} \\ & \det \begin{bmatrix} \delta_{i_1 j_1} & \delta_{i_1 j_2} & \cdots & \delta_{i_1 j_{N-1}} \\ \delta_{i_2 j_1} & \delta_{i_2 j_2} & \cdots & \delta_{i_2 j_{N-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{i_{N-1} j_1} & \delta_{i_{N-1} j_2} & \cdots & \delta_{i_{N-1} j_{N-1}} \end{bmatrix} , \end{aligned} \tag{8}$$

where the Kronecker delta, δ_{ij}, is defined as

$$\begin{aligned} \delta_{ij} \quad &= \quad 1 \qquad\qquad i = j \\ &= \quad 0 \qquad\qquad i \neq j \; . \end{aligned}$$

It is easy to verify that for $N = 3$ this reduces to eq. (7).

More remarkable, however, is the fact that this formula shows that the square magnitude of the normal $\vec{N}$ of a hyperplane given in eq. (6) is the *subvolume* of the corresponding parallelepiped specified by eq. (5). That is, not only does the *direction* of eq. (6) have an important geometric meaning with respect to the $(N-1)$-simplex specifying the hyperplane, but so does its *magnitude*! We find

$$\vec{N}\cdot\vec{N} = \det\begin{bmatrix} v(1,1) & v(1,2) & \cdots & v(1,N-1) \\ v(2,1) & v(2,2) & \cdots & v(2,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ v(N-1,1) & v(N-1,2) & \cdots & v(N-1,N-1) \end{bmatrix} = \left((N-1)!\, V_{N-1}\right)^2 .$$

♢ Clipping Tests in N Dimensions ♢

Now we can exploit the properties of the volume formula to define clipping ("which side") tests in any dimension. If we replace $(\vec{x}_N - \vec{x}_0)$ by $(\vec{x} - \vec{x}_0)$, eq. (3) becomes a *function* $V_N(\vec{x})$. Furthermore, this function has the remarkable property that it is an alternative form for the hyperplane equation, eq. (1), when $V_N(\vec{x}) = 0$.

We can furthermore determine *on which side* of the $(N-1)$-dimensional hyperplane determined by $(\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{N-1})$ an arbitrary point $\vec{x}$ lies simply by checking the sign of $V_N(\vec{x})$. That is,

- $V_N(\vec{x}) = 0 \Rightarrow$ the point $\vec{x}$ lies on a hyperplane and solves an equation of the form eq. (1).
- $V_N(\vec{x}) > 0 \Rightarrow$ the point $\vec{x}$ lies above the hyperplane.
- $V_N(\vec{x}) < 0 \Rightarrow$ the point $\vec{x}$ lies below the hyperplane.

Note: The special case $V_N = 0$ is of course just the general criterion for discovering *linear dependence* among a set of $(N+1)$ vector variables. This has the following elegant geometric interpretation: In 2D, we use the formula to compute the area of the triangle formed by 3 points $(\vec{x}_0, \vec{x}_1, \vec{x})$; if the area vanishes, the 3 points lie on a single line. In 3D, if the volume of the tetrahedron formed by 4 points $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x})$ vanishes, all 4 points are coplanar, and so on. Vanishing N-volume means the points lie in a hyperplane of dimension no greater than $(N-1)$.

These relationships between the sign of $V_N(\vec{x})$ and the relative position of $\vec{x}$ are precisely those we are accustomed to examining when we *clip* vectors (e.g., edges of a triangle) to lie on one side of a plane in a viewing frustum or within a projected viewing rectangle. For example, a 2D clipping line defined by the vector $\vec{x}_1 - \vec{x}_0 = (x_1 - x_0,\, y_1 - y_0)$ has a right-handed (unnormalized) normal $\vec{N} = (-(y_1 - y_0),\, (x_1 - x_0))$. Writing the 2D volume as the area A, eq. (3) becomes

$$A(\vec{x}) = \frac{1}{2}\det\begin{bmatrix} (x_1 - x_0) & (x - x_0) \\ (y_1 - y_0) & (y - y_0) \end{bmatrix} = \frac{1}{2}\left[\vec{N}\cdot(\vec{x} - \vec{x}_0)\right]$$

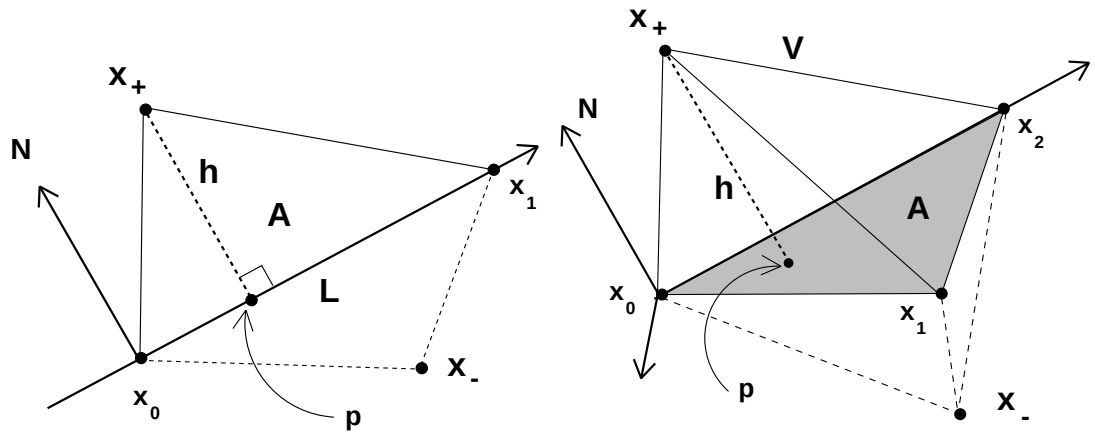


Figure 5. In 2D, the line through $\vec{x}_0$ to $\vec{x}_1$ defined by $\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = 0$ partitions the plane into two regions, one where this expression is positive (e.g., for $\vec{x}_+$) and another where it is negative (e.g., for $\vec{x}_-$). In 3D, the analogous procedure uses the plane defined by $(\vec{x}_0, \vec{x}_1, \vec{x}_2)$ to divide 3-space into two half spaces. The same pictures serve to show how the distance h from a point to a hyperplane is computable from the ratio of the simplex volume to the lower-dimensional volume of its base, i.e., $2A/L$ or $3V/A$.

for some arbitrary point $\vec{x}$, and so we recover the form of eq. (1) as

$$\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = \frac{2A}{\|\vec{x}_1 - \vec{x}_0\|} ,$$

where $\hat{\mathbf{n}} = \vec{N}/\|\vec{N}\|$; the relationship of $\vec{x}$ to the clipping line is determined by the sign.

In 3D, when clipping a line against a plane, everything reduces to the traditional form, namely the dot product between a 3D cross-product and a vector from a point $\vec{x}_0$ in the clipping plane to the point $\vec{x}$ being clipped. The normal to the plane through $(\vec{x}_0, \vec{x}_1, \vec{x}_2)$ is

$$\begin{aligned} \vec{N} &= (\vec{x}_1 - \vec{x}_0) \times (\vec{x}_2 - \vec{x}_0) \qquad (9) \\ &= \left(+\det \begin{bmatrix} (y_1 - y_0) & (y_2 - y_0) \\ (z_1 - z_0) & (z_2 - z_0) \end{bmatrix} , \right. \\ &\qquad \left. -\det \begin{bmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (z_1 - z_0) & (z_2 - z_0) \end{bmatrix} , +\det \begin{bmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{bmatrix} \right) , \end{aligned}$$

and we again find the same general form,

$$\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = \frac{6V}{\|\vec{N}\|} ,$$

whose sign determines where $\vec{x}$ falls. Figure 5 summarizes the relationship of the signed volume to the clipping task in 2D and 3D.

Hyperplanes for clipping applications in any dimension are therefore easily defined and checked by choosing $\vec{x}_N$ to be the test point $\vec{x}$ and checking the sign of eq. (3). If $\vec{N}$ and a point $\vec{x}_0$ are easy to determine directly, then the procedure reduces to checking the sign of the left hand side of eq. (1).

The final step is to find the desired point on the truncated, clipped line. Since the clipped form of a triangle, tetrahedron, etc., can be determined from the clipped forms of the component lines, we need only consider the point at which a line straddling the clipping hyperplane intersects this hyperplane. If the line to be clipped is given parametrically as $\vec{x}(t) = \vec{x}_a + t(\vec{x}_b - \vec{x}_a)$, where $\vec{x}_a$ and $\vec{x}_b$ are on opposite sides of the clipping hyperplane so $0 \le t \le 1$, then we simply plug $\vec{x}(t)$ into $V(\vec{x}) = 0$ and solve for t:

$$t = \frac{\det\left[\begin{array}{cccc} \vec{x}_1 - \vec{x}_0 & \vec{x}_2 - \vec{x}_0 & \cdots & \vec{x}_a - \vec{x}_0 \end{array}\right]}{\det\left[\begin{array}{cccc} \vec{x}_1 - \vec{x}_0 & \vec{x}_2 - \vec{x}_0 & \cdots & \vec{x}_a - \vec{x}_b \end{array}\right]} = \frac{\hat{\mathbf{n}} \cdot (\vec{x}_a - \vec{x}_0)}{\hat{\mathbf{n}} \cdot (\vec{x}_a - \vec{x}_b)} \,. \tag{10}$$

Here $\hat{\mathbf{n}}$ is of course just the normal to the clipping hyperplane, discussed in detail above.

♢ Point-Hyperplane Distance ♢

The general formula for the volume of a parallelepiped is the product of the base and the height, $W = Bh$. In N dimensions, if we take $W_N = N!\, V_N$ to be the volume of the parallelepiped with edges $(\vec{x}_1 - \vec{x}_0), (\vec{x}_2 - \vec{x}_0), \ldots, (\vec{x}_{N-1} - \vec{x}_0), (\vec{x} - \vec{x}_0)$, this generalizes to

$$W_N = h W_{N-1} \,,$$

where h is the perpendicular distance from the point $\vec{x}$ to the $(N-1)$-dimensional parallelepiped with volume $W_{N-1} = (N-1)!\, V_{N-1}$ and edges $(\vec{x}_1 - \vec{x}_0), (\vec{x}_2 - \vec{x}_0), \ldots, (\vec{x}_{N-1} - \vec{x}_0)$. We may thus immediately compute the distance h from a point to a hyperplane as

$$h = \frac{W_N}{W_{N-1}} = \frac{N!\, V_N}{(N-1)!\, V_{N-1}} = \frac{N\, V_N}{V_{N-1}} \,. \tag{11}$$

Note! Here one must use the trick of eq. 4 to express W_{N-1} in terms of the square root of a square determinant given by the product of two non-square matrices.

Thus in 2D, the area of a triangle $(\vec{x}_0, \vec{x}_1, \vec{x})$ is

$$A = V_2 = \frac{1}{2} W_2 = \frac{1}{2} \det\left[\begin{array}{cc} (x_1 - x_0) & (x - x_0) \\ (y_1 - y_0) & (y - y_0) \end{array}\right]$$

and the length-squared of the base is $L^2 = (\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0)$ so, with $A = (1/2)hL$, the height becomes $h = 2A/L = W_2/L = W_2/W_1$. In 3D, the volume of the tetrahedron $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x})$ is $V = V_3 = (1/6)W_3$ and the area $A = (1/2)W_2$ of the triangular base may be written

$$(2A)^2 = (W_2)^2 = \det\left[\begin{array}{cc} (\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0) & (\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_2 - \vec{x}_0) \\ (\vec{x}_2 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0) & (\vec{x}_2 - \vec{x}_0) \cdot (\vec{x}_2 - \vec{x}_0) \end{array}\right] \,.$$

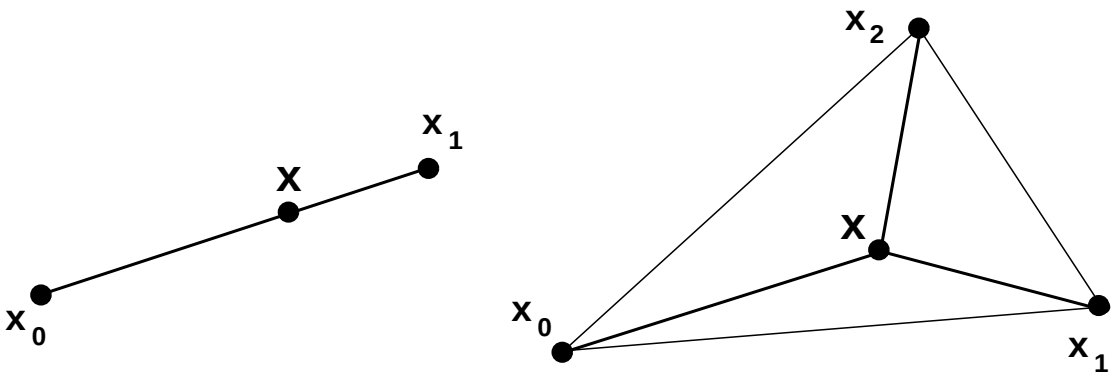


Figure 6. Barycentric coordinates in N dimensions.

We know $V = (1/3)hA$, and so $h = 3V/A = 6V/2A = W_3/W_2$. (See figure 5.) We note for reference that, as we showed earlier, the base $(N-1)$-volume is related to its normal by $\vec{N} \cdot \vec{N} = W_{N-1}^2$.

Here we also typically need to answer one last question, namely *where* is the point $\vec{p}$ on the base hyperplane closest to the point $\vec{x}$ whose distance h we just computed? This can be found by parameterizing the line from $\vec{x}$ to the base hyperplane along the normal $\hat{\mathbf{n}}$ to the hyperplane as $\vec{x}(t) = \vec{x} + t\hat{\mathbf{n}}$, writing the implicit equation for the hyperplane as $\hat{\mathbf{n}} \cdot (\vec{x}(t) - \vec{x}_0) = 0$, and solving for the mutual solution $t_p = \hat{\mathbf{n}} \cdot (\vec{x}_0 - \vec{x}) = -h$. Thus

$$\begin{aligned} \vec{p} &= \vec{x} + \hat{\mathbf{n}}(\hat{\mathbf{n}} \cdot (\vec{x}_0 - \vec{x})) \\ &= \vec{x} - h\hat{\mathbf{n}} \ . \end{aligned}$$

♢ Barycentric Coordinates ♢

Barycentric coordinates (see, e.g., (Hocking and Young 1961), chapter 5) are a practical way to parameterize lines, surfaces, etc., for applications that must compute where various geometric objects intersect. In practice, the barycentric coordinate method reduces to specifying two points $(\vec{x}_0, \vec{x}_1)$ on a line, three points $(\vec{x}_0, \vec{x}_1, \vec{x}_2)$ on a plane, four points $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \vec{x}_3)$ in a volume, etc., and parameterizing the line segment, enclosed triangular area, and enclosed tetrahedral volume, etc., respectively, by

$$\vec{x}(t) = \vec{x}_0 + t(\vec{x}_1 - \vec{x}_0) \tag{12}$$

$$\vec{x}(t_1, t_2) = \vec{x}_0 + t_1(\vec{x}_1 - \vec{x}_0) + t_2(\vec{x}_2 - \vec{x}_0) \tag{13}$$

$$\vec{x}(t_1, t_2, t_3) = \vec{x}_0 + t_1(\vec{x}_1 - \vec{x}_0) + t_2(\vec{x}_2 - \vec{x}_0) + t_3(\vec{x}_3 - \vec{x}_0) \tag{14}$$

$$\cdots \ .$$

The line and plane geometries are shown in figure 6. The interpolated point then lies within the N-simplex defined by the specified points provided

$$0 \le t \le 1$$
$$0 \le t_1 \le 1,\ 0 \le t_2 \le 1,\ 0 \le (1 - t_1 - t_2) \le 1$$
$$0 \le t_1 \le 1,\ 0 \le t_2 \le 1,\ 0 \le t_3 \le 1,\ 0 \le (1 - t_1 - t_2 - t_3) \le 1$$
$$\ldots$$

Center of What? However, this is really only half the story of barycentric coordinates. For the other half, we seek a geometric interpretation of the parameters t_i when we are *given* the value of $\vec{x}$.

First let us look at the simple case when $\vec{x}$ lies on the line segment between $\vec{x}_0$ and $\vec{x}_1$. Solving eq. (12) for t directly gives

$$t = \frac{(\vec{x} - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0)}{(\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0)} \ .$$

That is, t is the fraction of the distance that $\vec{x}$ has traveled along the line, the *ratio* between the length from $\vec{x}_0$ to $\vec{x}$ and the total length. But, since $\vec{x}_1 - \vec{x}_0 = \vec{x}_1 - \vec{x} + \vec{x} - \vec{x}_0$, we easily see that an alternative parameterization would be to take $t_1 = t$ and

$$t_0 = \frac{(\vec{x}_1 - \vec{x}) \cdot (\vec{x}_1 - \vec{x}_0)}{(\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0)}$$

so that $1 = t_0 + t_1$ and eq. (12) for $\vec{x}$ becomes

$$\vec{x}(t_0, t_1) = t_0 \vec{x}_0 + t_1 \vec{x}_1 \ .$$

If $t_0 = 1$, then the entire fraction of the distance from $\vec{x}_1$ to $\vec{x}$ is assigned to t_0 and $\vec{x} = \vec{x}_0$. If $t_1 = 1$, then the entire fraction of the distance from $\vec{x}_0$ to $\vec{x}$ is assigned to t_1 and $\vec{x} = \vec{x}_1$.

Next, suppose we know $\vec{x}$ in a plane and wish to compute its barycentric coordinates by solving eq. (13) for $(t_1,\ t_2)$. Once we realize that $(\vec{x}_1 - \vec{x}_0)$ and $(\vec{x}_2 - \vec{x}_0)$ form the basis for an affine coordinate system for the plane specified by $(\vec{x}_0, \vec{x}_1, \vec{x}_2)$ in *any* dimension, we see that we may measure the relative barycentric coordinates by taking the dot product with each basis vector:

$$\begin{aligned}(\vec{x} - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0) &= t_1 \|\vec{x}_1 - \vec{x}_0\|^2 + t_2 (\vec{x}_2 - \vec{x}_0) \cdot (\vec{x}_1 - \vec{x}_0) \\ (\vec{x} - \vec{x}_0) \cdot (\vec{x}_2 - \vec{x}_0) &= t_1 (\vec{x}_1 - \vec{x}_0) \cdot (\vec{x}_2 - \vec{x}_0) + t_2 \|\vec{x}_2 - \vec{x}_0\|^2 \ .\end{aligned}$$

Extending the previously introduced abbreviation to the form $v(x, j) = (\vec{x} - \vec{x}_0) \cdot (\vec{x}_j - \vec{x}_0)$ and solving this pair of equations by Cramer's rule, we get

$$t_1 = \frac{\det \begin{bmatrix} v(x,1) & v(1,2) \\ v(x,2) & v(2,2) \end{bmatrix}}{\det \begin{bmatrix} v(1,1) & v(1,2) \\ v(2,1) & v(2,2) \end{bmatrix}}$$

$$t_2 \;=\; \frac{\det\begin{bmatrix} v(1,1) & v(x,1) \\ v(1,2) & v(x,2) \end{bmatrix}}{\det\begin{bmatrix} v(1,1) & v(1,2) \\ v(2,1) & v(2,2) \end{bmatrix}} \,.$$

The denominator is clearly proportional to the *square* of the area of the triangle $(\vec{x}_0, \vec{x}_1, \vec{x}_2)$, and the numerators have the form of squared areas as well. In N dimensions, the numerators reduce to determinants of products of non-square matrices, and so may *not* be expressed as two separate determinants! However, if we transform to a coordinate system that contains the triangle within the plane of two coordinate axes, or if $N = 2$, an effectively square matrix is recovered; one factor of area in the denominator then cancels out, giving the intuitively expected result that the barycentric coordinates are ratios of two areas: $t_1 = A(\vec{x}, \vec{x}_0, \vec{x}_1)/A(\vec{x}_0, \vec{x}_1, \vec{x}_2)$, $t_2 = A(\vec{x}, \vec{x}_2, \vec{x}_0)/A(\vec{x}_0, \vec{x}_1, \vec{x}_2)$. This leads us to introduce the generalized version of t_0 for the line, namely,

$$\begin{aligned} t_0 \;&=\; 1 - t_1 - t_2 = \frac{A(\vec{x}_1, \vec{x}_2, \vec{x})}{A(\vec{x}_0, \vec{x}_1, \vec{x}_2)} \\ &=\; \frac{\det\begin{bmatrix} (\vec{x}_1 - \vec{x}_0)\cdot(\vec{x}_1 - \vec{x}) & (\vec{x}_1 - \vec{x}_0)\cdot(\vec{x}_2 - \vec{x}) \\ (\vec{x}_2 - \vec{x}_0)\cdot(\vec{x}_1 - \vec{x}) & (\vec{x}_2 - \vec{x}_0)\cdot(\vec{x}_2 - \vec{x}) \end{bmatrix}}{\det\begin{bmatrix} v(1,1) & v(1,2) \\ v(2,2) & v(2,2) \end{bmatrix}} \,. \end{aligned}$$

Here we used the squaring argument given above to extend t_0 from its special-coordinate-system interpretation as the fraction of the area contributed by the triangle $(\vec{x}, \vec{x}_1, \vec{x}_2)$ to the invariant form. This form obviously has the desired property that $t_0 = 1$ when $\vec{x} = \vec{x}_0$, and we finally have the sought equation (with $1 = t_0 + t_1 + t_2$)

$$\vec{x}(t_0, t_1, t_2) = t_0\vec{x}_0 + t_1\vec{x}_1 + t_2\vec{x}_2 \,.$$

It is amusing to note that the determinant identity $1 = t_0 + t_1 + t_2$ and its higher analogs, which are nontrivial to derive, generalize the simple identity $\vec{x}_1 - \vec{x}_0 = \vec{x}_1 - \vec{x} + \vec{x} - \vec{x}_0$ that we used in the 1D case.

Thus we can construct barycentric coordinates in any dimension which intuitively correspond to *fractions of hypervolumes*; each barycentric coordinate is the hypervolume of an N-simplex defined by the point $\vec{x}$ and all but one of the other simplex-defining points divided by the volume of the whole simplex. The actual computation, however, is best done using the squared-volume form because only that form is independent of the chosen coordinate system.

Note: The volumes are *signed*; even if $\vec{x}$ lies outside the N-simplex volume, the ratios remain correct due to the cancellation between the larger volumes and the negative volumes. We also remark that the generalized formulas for t_i in any dimension, with $1 = \sum_{i=0}^{N} t_i$, give an elegant geometric interpretation of Cramer's rule as ratios of simplex volumes.

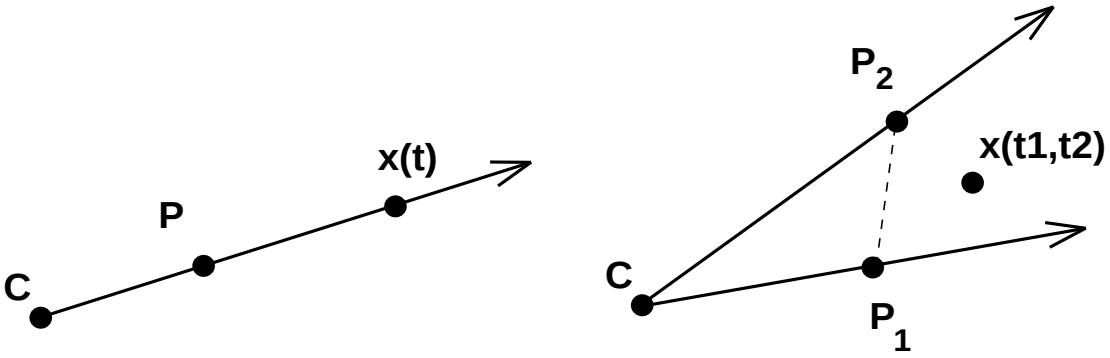


Figure 7. Schematic diagram comparing an ordinary camera ray and a planar "thick ray" used in N-dimensional ray-tracing methods.

♢ Ray Tracing ♢

It is often useful to compute the intersection of a ray passing through two points (typically the camera focal point $\vec{C}$ and an image point $\vec{P}$) with a geometrical object. In N dimensions, this object will typically be an $(N-1)$-simplex defining an oriented visible "face" with a normal vector computable as described above. We need to do several things: compute the intersection of the ray with the hyperplane containing the "face," check to see whether the point lies within the simplex's boundaries (observe that this is a clipping problem), and see whether the normal vector points in the direction of the ray (making it visible).

We formulate this procedure by first writing

$$\vec{X}(t) = \vec{C} + t(\vec{P} - \vec{C})$$

for the position of a point on the camera ray, as illustrated in figure 7. Then we consider a single $(N-1)$-simplex of the tessellation to be described either by a known normal or by using the set of N points giving its vertices to define its normal via eq. (6); in either case, we can write the equation of any *other* point $\vec{x}$ lying within the simplex as

$$\hat{\mathbf{n}} \cdot (\vec{x} - \vec{x}_0) = 0 \; .$$

Plugging in the parametric ray equation, we solve for the point $\vec{X}(t)$ in the simplex that lies on the ray:

$$t = \frac{\hat{\mathbf{n}} \cdot \left(\vec{x}_0 - \vec{C}\right)}{\hat{\mathbf{n}} \cdot \left(\vec{P} - \vec{C}\right)} \; .$$

A useful generalization of ray-tracing to N-dimensions follows from the observation that a "thick ray" is cast into space by an open-ended simplex that is essentially a barycentric coordinate form with the restriction $0 \leq (1 - t_1 - t_2 - \ldots) \leq 1$ relaxed (see, e.g., (Hanson and Cross 1993)).

A planar ray such as that shown in figure 7 then has two parameters,

$$\vec{X}(t_1, t_2) = \vec{C} + t_1(\vec{P}_1 - \vec{C}) + t_2(\vec{P}_2 - \vec{C}) \,,$$

with obvious generalizations to volume rays, etc. Intersecting such a planar ray with an $(N-2)$-dimensional manifold (describable using $(N-2)$ barycentric parameters) results in N equations with N unknown parameters, and thus a unique *point* is determined as the mutual solution. In 3D, a plane intersects a line in one point, in 4D two planes intersect in a single point, while in 5D a plane intersects a volume in a point. Other generalizations, including rays that intersect particular geometries in lines and surfaces, can easily be constructed. For example, the intersection of a planar ray with the single hyperplane equation for a 3-manifold in 4D leaves one undetermined parameter, and is therefore a line.

♢ Conclusion ♢

Geometry is an essential tool employed in the creation of computer graphics images of everyday objects. Statistical data analysis, mathematics, and science, on the other hand, provide many problems where N-dimensional generalizations of the familiar 2D and 3D formulas are required. The N-dimensional formulas and insights into the nature of geometry that we have presented here provide a practical guide for extending computer graphics into these higher-dimensional domains.

♢ Appendix: Determinants and the Levi-Civita Symbol ♢

One of the unifying features that has permitted us throughout this treatment to extend formulas to arbitrary dimensions has been the use of *determinants*. But what if you encounter an expression involving determinants that has not been given here and you wish to work out its algebraic properties for yourself? In this appendix, we outline for the reader a useful mathematical tool for treating determinants, the Levi-Civita symbol. References for this are hard to locate; the author learned these techniques by apprenticeship while studying general relativity, but even classic texts like Møller (Møller 1972) contain only passing mention of the methods; somewhat more detail is given in hard-to-find sources such as (Efimov and Rozendorn 1975).

First we define two basic objects, the Kronecker delta, δ_{ij},

$$\begin{aligned} \delta_{ij} &= 1 \qquad i = j \\ &= 0 \qquad i \neq j \end{aligned}$$

and the Levi-Civita symbol, $\epsilon_{ijk\ldots}$, which is the totally antisymmetric pseudotensor with the properties

$$\begin{aligned} \epsilon_{ijk\ldots} &= 1 \qquad i, j, k, \ldots \text{ in an even permutation of cyclic order} \\ &= -1 \qquad i, j, k, \ldots \text{ in an odd permutation of cyclic order} \\ &= 0 \qquad \text{when any two indices are equal.} \end{aligned}$$

All indices are assumed to range from 1 to N, e.g., $i = \{1, 2, \ldots, (N-1), N\}$, so that, for example, (1234,1342,4132,4321), are even permutations and (1324,2134,1243,4312) are odd permutations.

We can use the Kronecker delta to write the dot product between two N-dimensional vectors as a matrix product with the Kronecker delta representing the unit matrix,

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{N} \sum_{j=1}^{N} A_i \delta_{ij} B_j = \sum_{i=1}^{N} A_i \left(\sum_{j=1}^{N} \delta_{ij} B_j \right) = \sum_{i=1}^{N} A_i B_i \, , \tag{15}$$

and the Levi-Civita symbol to write the determinant of a matrix $|M|$ as

$$\det [M] \quad = \quad \sum_{\text{all } i_k \text{ indices}} \epsilon_{i_1 i_2 \ldots i_N} M_{1,i_1} M_{2,i_2} \cdots M_{N,i_N} \, .$$

The fundamental formula for the product of two Levi-Civita symbols is:

$$\epsilon_{i_1 i_2 \ldots i_N} \epsilon_{j_1 j_2 \ldots j_N} \quad = \quad \det \begin{bmatrix} \delta_{i_1 j_1} & \delta_{i_1 j_2} & \cdots & \delta_{i_1 j_N} \\ \delta_{i_2 j_1} & \delta_{i_2 j_2} & \cdots & \delta_{i_2 j_N} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{i_N j_1} & \delta_{i_N j_2} & \cdots & \delta_{i_N j_N} \end{bmatrix} \, .$$

(Note that if we set $\{j_1 j_2 \ldots j_N\} = \{1, 2, \ldots, N\}$, the second Levi-Civita symbol reduces to $+1$, and the resulting determinant is an explicit realization of the antisymmetry of the Levi-Civita symbol itself as a determinant of Kronecker deltas!)

With this notation, the generalized cross product $\vec{N}$ of eq. (6), simplified by setting $\vec{x}_0 = 0$, can be written

$$\vec{N} \quad = \quad \sum_{\text{all indices}} \epsilon_{i_1 i_2 \ldots i_{N-1} i_N} x_1^{(i_1)} x_2^{(i_2)} \cdots x_{N-1}^{(i_{N-1})} \hat{\mathbf{x}}^{(i_N)} \, ,$$

where $\hat{\mathbf{x}}^{(i_N)}$ are the unit vectors $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \ldots, \hat{\mathbf{w}})$ of the coordinate system. The dot product between the normal and another vector simply becomes

$$\vec{N} \cdot \vec{L} \quad = \quad \sum_{\text{all indices}} \epsilon_{i_1 i_2 i_3 \ldots i_{N-1} i_N} x_1^{(i_1)} x_2^{(i_2)} x_3^{(i_3)} \cdots x_{N-1}^{(i_{N-1})} \, L^{(i_N)} \, .$$

The reader can verify that, in 2D, $N_k = \sum_{i=1}^{2} x^{(i)} \epsilon_{ik} = (-y, +x)$, and so on. We conclude with two examples of applications:

Rotations of Normals. Is the normal $\vec{N}$ a vector? *Almost*. To check this, we must rotate each column vector in the cross product formula using $x'^{(i)} = \sum_{j=1}^{N} R_{ij} x^{(j)}$ and compute the behavior of $\vec{N}$. Using the identity ((Efimov and Rozendorn 1975), p. 203),

$$\epsilon_{i_1 i_2 \ldots i_{N-1} i_N} \det[R] = \sum_{\text{all } j_k \text{ indices}} \epsilon_{j_1 j_2 \ldots j_{N-1} j_N} R_{j_1 i_1} R_{j_2 i_2} \cdots R_{j_{N-1} i_{N-1}} R_{j_N i_N} ,$$

we find

$$\begin{aligned} N'^{(i)} &= \sum_{\substack{\text{all indices} \\ \text{except } i}} \epsilon_{i_1 i_2 \ldots i_{N-1} i} R_{i_1 j_1} x_1^{(j_1)} R_{i_2 j_2} x_2^{(j_2)} \cdots R_{i_{N-1} j_{N-1}} x_{N-1}^{(j_{N-1})} \\ &= \sum_{j=1}^{N} R_{ij} N^{(j)} \det[R] . \end{aligned}$$

Therefore $\vec{N}$ is a *pseudotensor*, and behaves as a vector for ordinary rotations (which have $\det[R] = 1$), but changes sign if $[R]$ contains an odd number of reflections.

Contraction Formula. The contraction of two partial determinants of $(N-K)$ N-dimensional vectors can expanded in terms of products of Kronecker deltas as follows:

$$\sum_{i_{N-K+1} \ldots i_N} \epsilon_{i_1 i_2 \ldots i_{N-K} i_{N-K+1} \ldots i_N} \epsilon_{j_1 j_2 \ldots j_{N-K} i_{N-K+1} \ldots i_N} = K! \det \begin{bmatrix} \delta_{i_1 j_1} & \delta_{i_1 j_2} & \cdots & \delta_{i_1 j_{N-K}} \\ \delta_{i_2 j_1} & \delta_{i_2 j_2} & \cdots & \delta_{i_2 j_{N-K}} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{i_{N-K} j_1} & \delta_{i_{N-K} j_2} & \cdots & \delta_{i_{N-K} j_{N-K}} \end{bmatrix} .$$

The expression eq. (8) for the dot product of two normals is a special case of this formula.

Acknowledgment. This work was supported in part by NSF grant IRI-91-06389.

Bibliography

(Banchoff and Werner 1983) T. Banchoff and J. Werner. *Linear Algebra through Geometry*. Springer-Verlag, 1983.

(Banchoff 1990) Thomas F. Banchoff. *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions*. Scientific American Library, New York, NY, 1990.

(Brun et al. 1989) R. Brun, O. Couet, C. Vandoni, and P. Zanarini. *PAW – Physics Analysis Workstation, The Complete Reference*. CERN, Geneva, Switzerland, October 1989. Version 1.07.

(Chen et al. 1988) Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-d rotation using 2-d control devices. In *Proceedings of Siggraph 88*, volume 22, pages 121–130, 1988.

(Coxeter 1991) H.S.M. Coxeter. *Regular Complex Polytopes*. Cambridge University Press, second edition, 1991.

(Edmonds 1957) A.R. Edmonds. *Angular Momentum in Quantum Mechanics*. Princeton University Press, Princeton, New Jersey, 1957.

(Efimov and Rozendorn 1975) N.V. Efimov and E.R. Rozendorn. *Linear Algebra and Multi-Dimensional Geometry*. Mir Publishers, Moscow, 1975.

(Feiner and Beshers 1990a) S. Feiner and C. Beshers. Visualizing n-dimensional virtual worlds with n-vision. *Computer Graphics*, 24(2):37–38, March 1990.

(Feiner and Beshers 1990b) S. Feiner and C. Beshers. Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds. In *Proceedings of UIST '90, Snowbird, Utah*, pages 76–83, October 1990.

(Francis 1987) G.K. Francis. *A Topological Picturebook*. Springer-Verlag, New York, 1987.

(Hanson and Cross 1993) A.J. Hanson and R.A. Cross. Interactive visualization methods for four dimensions. In *IEEE Visualization '93*, 1993. To be published.

ISBN 0-12-336156-7

(Hanson and Heng 1992a) Andrew J. Hanson and Pheng A. Heng. Four-dimensional views of 3-D scalar fields. In Arie E. Kaufman and Gregory M. Nielson, editors, *Proceedings of Visualization 92*, pages 84–91, Los Alamitos, CA, October 1992. IEEE Computer Society Press.

(Hanson and Heng 1992b) Andrew J. Hanson and Pheng A. Heng. Illuminating the fourth dimension. *IEEE Computer Graphics and Applications*, 12(4):54–62, July 1992.

(Hanson 1992) Andrew J. Hanson. The rolling ball. In David Kirk, editor, *Graphics Gems III*, pages 51–60. Academic Press, San Diego, CA, 1992.

(Hocking and Young 1961) John G. Hocking and Gail S. Young. *Topology*. Addison-Wesley, 1961.

(Møller 1972) C. Møller. *The Theory of Relativity*. Oxford, Clarendon Press, 1972.

(Noll 1967) Michael A. Noll. A computer technique for displaying n-dimensional hyperobjects. *Communications of the ACM*, 10(8):469–473, August 1967.

(Phillips et al. 1993) Mark Phillips, Silvio Levy, and Tamara Munzner. Geomview: An interactive geometry viewer. *Notices of the Amer. Math. Society*, 40(8):985–988, October 1993. Available by anonymous ftp from geom.umn.edu, The Geometry Center, Minneapolis MN.

(Sommerville 1958) D.M.Y. Sommerville. *An Introduction to the Geometry of N Dimensions*. Reprinted by Dover Press, 1958.

◊ II.4

Rotations for N-Dimensional Graphics

Andrew J. Hanson
Computer Science Department
Indiana University
Bloomington, IN 47405
hanson@cs.indiana.edu

◊ Introduction ◊

In a previous Gem (Hanson 1994), "Geometry for N-Dimensional Graphics," we described a family of techniques for dealing with the geometry of N-dimensional models in the context of graphics applications. Here, we build on that framework to look in more detail at rotations in N-dimensional Euclidean space. In particular, we give a natural N-dimensional extension of the 3D rolling ball technique described in an earlier Gem (Hanson 1992), along with the corresponding analog of the Virtual Sphere method (Chen et al. 1988). Next, we touch on practical methods for specifying and understanding the parameters of N-dimensional rotations. Finally, we give the explicit 4D extension of 3D quaternion orientation splines.

For additional details and insights, we refer the reader to classic sources such as (Sommerville 1958,Coxeter 1991,Hocking and Young 1961,Efimov and Rozendorn 1975).

◊ The Rolling Ball in N Dimensions ◊

Basic Intuition of the Rolling Ball. The basic intuitive property of a rolling ball (or *tangent space*) rotation algorithm in any dimension is that it takes a unit vector $\hat{\mathbf{v}}_0 = (0, 0, \ldots, 0, 1)$ pointing purely in the N-th direction (towards the "north pole" of the ball) and tips it in the direction of an arbitrary unit vector $\hat{\mathbf{n}} = (n_1, n_2, \ldots, n_{N-1}, 0)$ lying in the $(N-1)$-plane tangent to the ball at the north pole, thus producing a new, rotated unit vector $\hat{\mathbf{v}}$, where

$$\hat{\mathbf{v}} = M_N \cdot \hat{\mathbf{v}}_0 = \hat{\mathbf{n}} \sin\theta + \hat{\mathbf{v}}_0 \cos\theta \ ,$$

as indicated schematically in Figure 1a. (Note: for notational simplicity, we choose to write the components of column vectors as horizontal lists.)

If we choose the convention that positive rotations are right-handed and progress counterclockwise, a positive rotation of the north pole actually tilts it into the negative direction of the remaining axis of the rotation plane. That is, if the 2D "rolling circle" acts on $\hat{\mathbf{v}}_0 = (0, 1)$ and

ISBN 0-12-543457-X

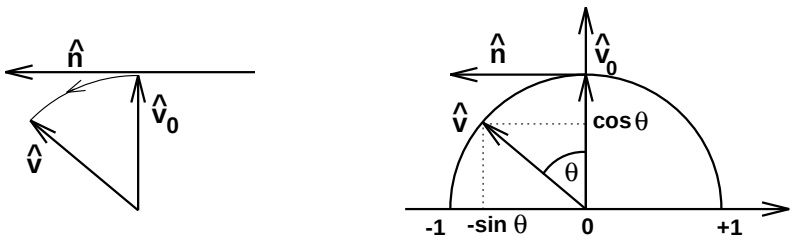


Figure 1. Tilting the "north pole" vector $\hat{\mathbf{v}}_0$ in the direction of the tangent vector $\hat{\mathbf{n}}$, as though rolling a ball by placing one's finger directly on the north pole and pulling in the direction $\hat{\mathbf{n}}$.

$\hat{\mathbf{n}} = (-1, 0)$ as shown in Figure 1b, then

$$\hat{\mathbf{v}} = M_2 \cdot \hat{\mathbf{v}}_0 = \hat{\mathbf{n}} \sin\theta + \hat{\mathbf{v}}_0 \cos\theta = (-\sin\theta,\, \cos\theta)\,,$$

where the rotation matrix M_2 can be written

$$\begin{aligned} M_2 &= \begin{bmatrix} \cos\theta & -\sin\theta \\ +\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} c & -s \\ +s & c \end{bmatrix} \\ &= \begin{bmatrix} c & +n_x s \\ -n_x s & c \end{bmatrix}\,. \end{aligned} \tag{1}$$

If we choose a right-handed overall coordinate frame, the sign of $\hat{\mathbf{n}}$ will automatically generate the correct sign convention.

Synopsis: Qualitatively speaking, if we imagine looking straight down at the north pole, the rolling ball *pulls* the unseen N-th component of a vector along the direction $\hat{\mathbf{n}}$ of the $(N-1)$-dimensional controller motion, bringing the unseen component gradually into view.

Implementation. In practice, we choose a radius R for the ball containing the object or scene to be rotated and move our controller (slider, 2D mouse, 3D mouse, ...) a distance r in the tangent direction $\hat{\mathbf{n}}$, as indicated in Figure 2a. Working from the simplified diagram in Figure 2b, we define $D^2 = R^2 + r^2$ and choose the rotation parameters $c = \cos\theta = R/D$ and $s = \sin\theta = r/D$.

For interactive systems, this choice has the particular advantage that, however rapidly the user moves the controller, $0 \le (r/D) < +1$, so $0 \le \theta < \pi/2$. Depending upon the desired interface behavior, an alternative choice would be to take $\theta = r/R$. This requires computing a trigonometric function instead of a square root, and may cause large discontinuities in orientation for large controller motion.

3D. The explicit 3D rolling ball formula can be derived starting from an arbitrary 2D mouse displacement $\vec{\mathbf{r}} = (x, y, 0) = (rn_x, rn_y, 0)$, where $n_x^2 + n_y^2 = 1$. Then one replaces Equation

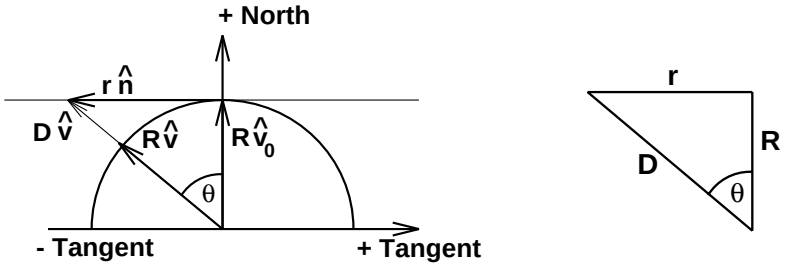


Figure 2. The notation used in implementing the rolling ball rotation model for N dimensions.

(1) with $n_x = +1$ by the analogous 3×3 matrix R_0 for (x, z) rotations and encloses this in a conjugate pair of rotations R_{xy} that transform the 2D mouse displacement $\vec{\mathbf{r}}$ into the strictly positive x-direction and back. Since even $\vec{\mathbf{r}} = (-1, 0, 0)$ is rotated to the positive x-direction before R_0 acts, all signs are correct. With the explicit matrices

$$R_{xy} = \begin{bmatrix} n_x & -n_y & 0 \\ n_y & n_x & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; R_0 = \begin{bmatrix} c & 0 & +s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix},$$

we find an alternative derivation of the formula in our earlier Gem (Hanson 1992):

$$\begin{aligned} M_3 &= R_{xy} R_0 (R_{xy})^{-1} \\ &= \begin{bmatrix} c + (n_y)^2(1-c) & -n_x n_y (1-c) & n_x s \\ -n_x n_y (1-c) & c + (n_x)^2(1-c) & n_y s \\ -n_x s & -n_y s & c \end{bmatrix} \\ &= \begin{bmatrix} 1 - (n_x)^2(1-c) & -n_x n_y (1-c) & n_x s \\ -n_x n_y (1-c) & 1 - (n_y)^2(1-c) & n_y s \\ -n_x s & -n_y s & c \end{bmatrix}. \end{aligned} \tag{2}$$

4D. The 4D case takes as input a 3D mouse motion $\vec{\mathbf{r}} = (x, y, z, 0) = (rn_x, rn_y, rn_z, 0)$, with $n_x^2 + n_y^2 + n_z^2 = 1$. Then one first transforms (n_y, n_z) into a pure y-component, rotates that result to yield a pure x-component, performs a rotation by θ in the (x, w)-plane, and reverses the first two rotations. Defining the required matrices as

$$R_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{n_y}{r_{yz}} & -\frac{n_z}{r_{yz}} & 0 \\ 0 & \frac{n_z}{r_{yz}} & \frac{n_y}{r_{yz}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_{xy} = \begin{bmatrix} n_x & -r_{yz} & 0 & 0 \\ r_{yz} & n_x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_0 = \begin{bmatrix} c & 0 & 0 & +s \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s & 0 & 0 & c \end{bmatrix}, \tag{3}$$

where $r_{yz}^2 = n_y^2 + n_z^2$, we find

$$\begin{aligned} M_4 &= R_{yz} R_{xy} R_0 (R_{xy})^{-1} (R_{yz})^{-1} \\ &= \begin{bmatrix} 1-(n_x)^2(1-c) & -(1-c)n_x n_y & -(1-c)n_x n_z & s n_x \\ -(1-c)n_x n_y & 1-(n_y)^2(1-c) & -(1-c)n_y n_z & s n_y \\ -(1-c)n_x n_z & -(1-c)n_y n_z & 1-(n_z)^2(1-c) & s n_z \\ -s n_x & -s n_y & -s n_z & c \end{bmatrix} . \end{aligned} \tag{4}$$

ND. The extension of this procedure to any dimension is accomplished by having the controller interface supply an $(N-1)$-dimensional vector $\vec{\mathbf{r}} = (r n_1, r n_2, \ldots, r n_{N-1}, 0)$ with $\vec{\mathbf{r}} \cdot \vec{\mathbf{r}} = r^2$ and $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$ and applying the rotation

$$\begin{aligned} M_N &= R_{N-2,N-1} R_{N-3,N-2} \cdots R_{1,2} R_0 (R_{1,2})^{-1} \cdots (R_{N-3,N-2})^{-1} (R_{N-2,N-1})^{-1} \\ &= \begin{bmatrix} 1-(n_1)^2(1-c) & -(1-c)n_2 n_1 & \cdots & -(1-c)n_{N-1} n_1 & s n_1 \\ -(1-c)n_1 n_2 & 1-(n_2)^2(1-c) & \cdots & -(1-c)n_{N-1} n_2 & s n_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -(1-c)n_1 n_{N-1} & -(1-c)n_2 n_{N-1} & \cdots & 1-(n_{N-1})^2(1-c) & s n_{N-1} \\ -s n_1 & -s n_2 & \cdots & -s n_{N-1} & c \end{bmatrix} \end{aligned} \tag{5}$$

Recall that the controller input $\vec{\mathbf{r}} = r\hat{\mathbf{n}}$ that selects the direction to "pull" also determines $c = \cos\theta = R/D$, $s = \sin\theta = r/D$, with $D^2 = R^2 + r^2$, or, alternatively, $\theta = r/R$.

♢ Controlling the Remaining Rotational Degrees of Freedom ♢

There are $N(N-1)/2$ parameters in a general N-dimensional orthogonal rotation matrix, one parameter for each possible pair of axes specifying a *plane of rotation* (the 3D intuition about "axes of rotation" does not extend simply to higher dimensions). The matrix M_N in Equation (5) has only $(N-1)$ independent parameters: we must now understand what happened to the other $(N-1)(N-2)/2$ degrees of freedom needed for arbitrary rotations.

In fact, the non-commutativity of the rotation group allows us to generate all the other rotations by *small circular motions* of the controller in the $(N-1)$-dimensional subspace of $\vec{\mathbf{r}} = r\hat{\mathbf{n}}$. Moving the controller in circles in the $(1,2)$-plane, $(1,3)$-plane, etc., of the $(N-1)$-dimensional controller space exactly generates the missing $(N-1)(N-2)/2$ rotations required to exhaust the full parameter space. In mathematical terms, the additional motions are generated by the commutation relations of the $SO(N)$ Lie algebra for $i, j = 1, \ldots, N-1$,

$$\begin{aligned} [R_{iN}, R_{jN}] &= \delta_{ij} R_{NN} - \delta_{jN} R_{iN} + \delta_{iN} R_{jN} - \delta_{NN} R_{ij} \\ &= -R_{ij} \ . \end{aligned}$$

The minus sign in the above equation means that *clockwise* controller motions in the (i, j)-plane inevitably produce *counterclockwise* rotations of the object, and vice-versa. Thus the philosophy (Hanson 1992) of achieving the full set of context-free rotation group transformations with a limited set of controller moves extends perfectly to N-dimensions. *Implementation Note*: In practice, the effectiveness of this technique varies considerably with the application; the size of the counter-rotation experienced may be relatively small for parameters that give appropriate spatial motion sensitivity with current 3D mouse technology.

Alternative Context Philosophies. The rolling ball interface is a *context-free* interface that allows the user of a virtual reality application to ignore the absolute position of the controller and requires no supplementary cursor context display; thus one may avoid distractions that may disturb stereography and immersive effects in a virtual reality environment. However some applications are better adapted to *context-sensitive* interfaces like the Arcball method (Shoemake 1994) or the Virtual Sphere approach (Chen et al. 1988). The Virtual Sphere approach in particular can be straightforwardly extended to higher dimensions by using the rolling ball equations inside a displayed spatial context (typically a sphere) and changing over to an $(N-1)$-dimensional rolling ball outside the context; that is, as the controller approaches and passes the displayed inner domain context sphere, the rotation action changes to one that leaves the N-th coordinate fixed but changes the remaining $(N-1)$ coordinates as though an $(N-1)$-dimensional rolling ball controller were attached to the nearest point on the sphere. Similar flexibility can be achieved by using a different controller state to signal a discrete rather than a continuous context switch to the $(N-1)$-dimensional controller.

♢ Handy Formulas for N-Dimensional Rotations ♢

For some applications the incremental orientation control methods described above are not as useful as knowing a single matrix for the entire N-dimensional orientation frame for an object. We note three ways to represent such an orientation frame:

Columns are new axes. One straightforward construction simply notes that if the default coordinate frame is represented by the orthonormal set of unit vectors $\hat{\mathbf{x}}_1 = (1, 0, \ldots, 0)$, $\hat{\mathbf{x}}_2 = (0, 1, 0, \ldots, 0)$, …, $\hat{\mathbf{x}}_N = (0, \ldots, 0, 1)$, and the desired axes of the new (orthonormal) coordinate frame are known to be $\hat{\mathbf{a}}_1 = (a_1^{(1)}, a_1^{(2)}, \ldots, a_1^{(N)})$, $\hat{\mathbf{a}}_2$, …, $\hat{\mathbf{a}}_N$, then the rotation matrix that transforms any vector to that frame just has the new axes as its columns:

$$M = \left[\begin{array}{cccc} \hat{\mathbf{a}}_1 & \hat{\mathbf{a}}_2 & \cdots & \hat{\mathbf{a}}_N \end{array} \right] .$$

The orthonormality constraints give M the required $N(N-1)/2$ degrees of freedom.

Concatenated subplane rotations. Rotations in the plane of a pair of coordinate axes $(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$, $i, j = 1, \ldots, N$ can be written as the block matrix

$$R_{ij}(\theta_{ij}) = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cos\theta_{ij} & 0 & \cdots & 0 & -\sin\theta_{ij} & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & \sin\theta_{ij} & 0 & \cdots & 0 & \cos\theta_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{6}$$

and thus the $N(N-1)/2$ distinct $R_{ij}(\theta_{ij})$ may be concatenated in some order to produce a rotation matrix such as

$$M = \prod_{i<j} R_{ij}(\theta_{ij})$$

with $N(N-1)/2$ degrees of freedom parametrized by $\{\theta_{ij}\}$. However, since the matrices R_{ij} do not commute, different orderings give different results and it is difficult to intuitively understand the global rotation. In fact, as is the case for 3D Euler angles, one may even repeat some matrices (with distinct parameters) and omit others, and still not miss any degrees of freedom.

Quotient Space Decomposition. Another useful decomposition relies on the classic quotient property of the topological spaces of the orthogonal groups (Helgason 1962),

$$SO(N)/SO(N-1) = S^{N-1}\,, \tag{7}$$

where S^K is a K-dimensional topological sphere. In practical terms, this means that the $N(N-1)/2$ parameters of $SO(N)$, the mathematical group of N-dimensional orthogonal rotations, can be viewed as a nested family of points on spheres. The 2D form is the matrix (1) parameterizing the points on the circle S^1; the 3D form reduces to the standard matrix

$$M_3(\theta, \hat{\mathbf{n}}) = \begin{bmatrix} c + (n_1)^2(1-c) & n_1 n_2(1-c) - s n_3 & n_3 n_1(1-c) + s n_2 \\ n_1 n_2(1-c) + s n_3 & c + (n_2)^2(1-c) & n_3 n_2(1-c) - s n_1 \\ n_1 n_3(1-c) - s n_2 & n_2 n_3(1-c) + s n_1 & c + (n_3)^2(1-c) \end{bmatrix} \tag{8}$$

where the two free parameters of $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = (n_1)^2 + (n_2)^2 + (n_3)^2 = 1$ describe a point on the 2-sphere. These two parameters plus a third from the S^1 described by $c^2 + s^2 = 1$ (i.e., $c = \cos\theta$, $s = \sin\theta$) yield the required total of three free parameters equivalent to the three Euler angles. The 4D and higher forms are already too unwieldy to be conveniently written as single matrices.

♢ Interpolating N-Dimensional Orientation Frames ♢

To define a uniform-angular-velocity interpolation between two N-dimensional orientation frames, we might consider independently interpolating each angle in Equation (6), or we might take the quotient space decomposition given by the hierarchy of points on the spheres $(S^{N-1}, \ldots, S^2, S^1)$ and apply a constant angular velocity spherical interpolation to each spherical point in each successive dimension using the "Slerp"

$$\hat{\mathbf{n}}_{12}(t) = \text{Slerp}(\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2, t) = \hat{\mathbf{n}}_1 \frac{\sin((1-t)\theta)}{\sin(\theta)} + \hat{\mathbf{n}}_2 \frac{\sin(t\theta)}{\sin(\theta)}$$

where $\cos\theta = \hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2$. (This formula is simply the result of applying a Gram-Schmidt decomposition while enforcing unit norm in any dimension.)

Either of these often achieves the goal of smooth appearance, but the solutions are neither unique nor mathematically compelling, since the curve is not guaranteed to be a geodesic in $SO(N)$.

The specification of geodesic curves in $SO(N)$ is a difficult problem in general (Barr et al. 1992); fortunately, the two most important cases for interactive systems, $N = 3$ and $N = 4$, have elegant solutions using the covering or "Spin" groups. For $SO(3)$, geodesic interpolations and suitable corresponding splines are definable using Shoemake's quaternion splines (Shoemake 1985), which can be simply formulated using Slerps on S^3 as follows: let $\hat{\mathbf{n}}$ be a unit 3-vector, so that

$$q_0 = \cos(\theta/2), \quad \vec{\mathbf{q}} = \hat{\mathbf{n}} \sin(\theta/2)$$

is automatically a point on S^3 due to the constraint $(q_0)^2 + (q_1)^2 + (q_2)^2 + (q_3)^2 = 1$. Then each point on S^3 corresponds to an $SO(3)$ rotation matrix

$$R_3 = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{bmatrix} \tag{9}$$

which the reader can verify reduces exactly to the nested-sphere form in Equation (8). Note that the quaternions q and $-q$ each correspond to the same 3D rotation. Slerping q generates sequences of matrices $R_3(t)$ that are geodesic interpolations. Arbitrary splines can be defined using the method of Schlag (Schlag 1991).

Quaternions in Four Dimensions. In four dimensions, the correspondence between the rotation group $SO(4)$ and the spin group $\text{Spin}(4)$ that is its double covering may be computed by extending quaternion multiplication to act not just on 3-vectors ("pure" quaternions) $v = (0, \vec{\mathbf{V}})$, but on full 4-vector quaternions v^μ in the following way:

$$\sum_{\nu=0}^{3} R^\mu_\nu v^\nu = q \cdot v^\mu \cdot p^{-1} .$$

We thus find that the general double-quaternion parameterization for 4D rotation matrices takes the form

$$R_4 = \begin{bmatrix} q_0p_0 + q_1p_1 + q_2p_2 + q_3p_3 & q_1p_0 - q_0p_1 - q_3p_2 + q_2p_3 \\ -q_1p_0 + q_0p_1 - q_3p_2 + q_2p_3 & q_0p_0 + q_1p_1 - q_2p_2 - q_3p_3 \\ -q_2p_0 + q_0p_2 - q_1p_3 + q_3p_1 & q_1p_2 + q_2p_1 + q_0p_3 + q_3p_0 \\ -q_3p_0 + q_0p_3 - q_2p_1 + q_1p_2 & q_1p_3 + q_3p_1 - q_0p_2 - q_2p_0 \end{bmatrix}$$
$$\begin{bmatrix} q_2p_0 - q_0p_2 - q_1p_3 + q_3p_1 & q_3p_0 - q_0p_3 - q_2p_1 + q_1p_2 \\ q_1p_2 + p_1q_2 - p_0q_3 - q_0p_3 & q_1p_3 + p_1q_3 + p_0q_2 + q_0p_2 \\ q_0p_0 + q_2p_2 - q_1p_1 - q_3p_3 & q_2p_3 + q_3p_2 - q_0p_1 - q_1p_0 \\ q_2p_3 + q_3p_2 + q_1p_0 + p_0q_1 & q_0p_0 + q_3p_3 - q_1p_1 - q_2p_2 \end{bmatrix} . \quad (10)$$

One may check that Equation (9) is just the lower right-hand corner of the degenerate $p = q$ case of Equation (10).

Shoemake-style interpolation between two distinct 4D frames is now achieved by applying the desired Slerp-based interpolation method independently to a set of quaternion coordinates $q(t)$ on one three-sphere, and to a separate set of quaternion coordinates $p(t)$ on another. The resulting matrix $R_4(t)$ gives geodesic interpolations for simple Slerps, and can be used as the basis for corresponding spline methods (Schlag 1991,Barr et al. 1992). Analogs of the $N = 3$ and $N = 4$ approaches for general N involve computing Spin(N) geodesics and thus are quite complex.

Controls. As pointed out in (Shoemake 1994), the Arcball controller can be adapted with complete faithfulness of spirit to the 4D case, since one can pick *two* points in a three-sphere to specify an initial 4D frame, and then pick *two more* points in the three-sphere to define the current 4D frame. Equation (10) gives the complete form of the effective 4D rotation. Alternately, one can replace the 4D rolling ball or Virtual Sphere controls described earlier by a pair (or more) of 3D controllers (Hanson 1992).

Acknowledgment. This work was supported in part by NSF grant IRI-91-06389.

♢ Bibliography ♢

(Barr et al. 1992) Alan H. Barr, Bena Currin, Steven Gabriel, and John F. Hughes. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics*, 26(2):313–320, 1992.

(Chen et al. 1988) Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of Siggraph 88*, volume 22, pages 121–130, 1988.

(Coxeter 1991) H.S.M. Coxeter. *Regular Complex Polytopes*. Cambridge University Press, second edition, 1991.

(Efimov and Rozendorn 1975) N.V. Efimov and E.R. Rozendorn. *Linear Algebra and Multi-Dimensional Geometry*. Mir Publishers, Moscow, 1975.

(Hanson 1992) Andrew J. Hanson. The rolling ball. In David Kirk, editor, *Graphics Gems III*, pages 51–60. Academic Press, 1992.

(Hanson 1994) Andrew J. Hanson. Geometry for n-dimensional graphics. In Paul Heckbert, editor, *Graphics Gems IV*, pages 149–170. Academic Press, 1994.

(Helgason 1962) Sigurdur Helgason. *Differential Geometry and Symmetric Spaces*. Academic Press, New York, 1962.

(Hocking and Young 1961) John G. Hocking and Gail S. Young. *Topology*. Addison-Wesley, 1961.

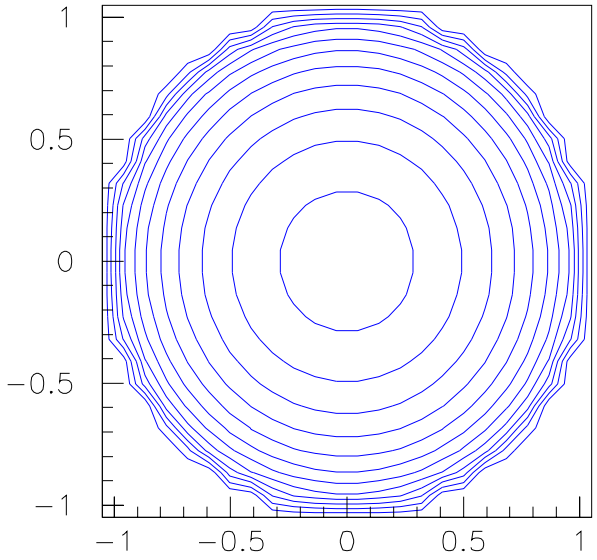
(Schlag 1991) John Schlag. Using geometric constructions to interpolate orientations with quaternions. In James Arvo, editor, *Graphics Gems II*, pages 377–380. Academic Press, 1991.

(Shoemake 1985) K. Shoemake. Animating rotation with quaternion curves. In *Computer Graphics*, volume 19, pages 245–254, 1985. Proceedings of SIGGRAPH 1985.

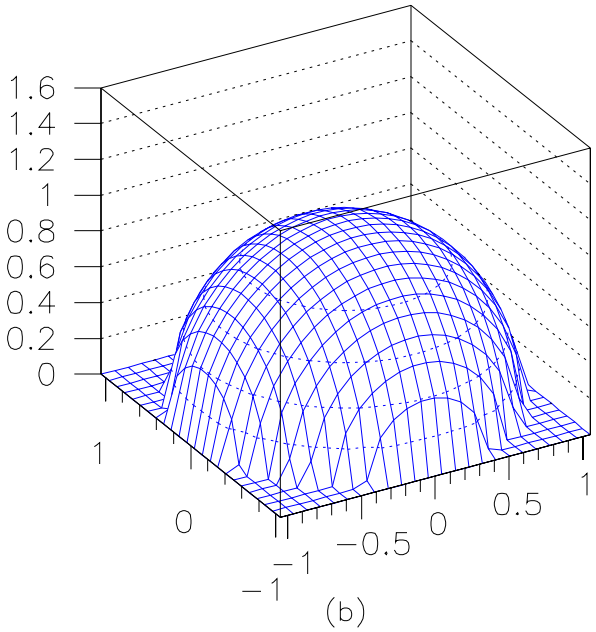
(Shoemake 1994) Ken Shoemake. Arcball rotation control. In Paul Heckbert, editor, *Graphics Gems IV*, pages 172–192. Academic Press, 1994.

(Sommerville 1958) D.M.Y. Sommerville. *An Introduction to the Geometry of N Dimensions*. Reprinted by Dover Press, 1958.

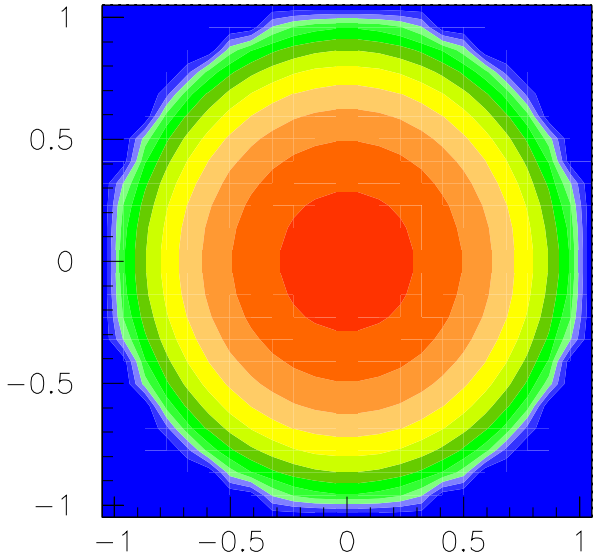
Hemisphere



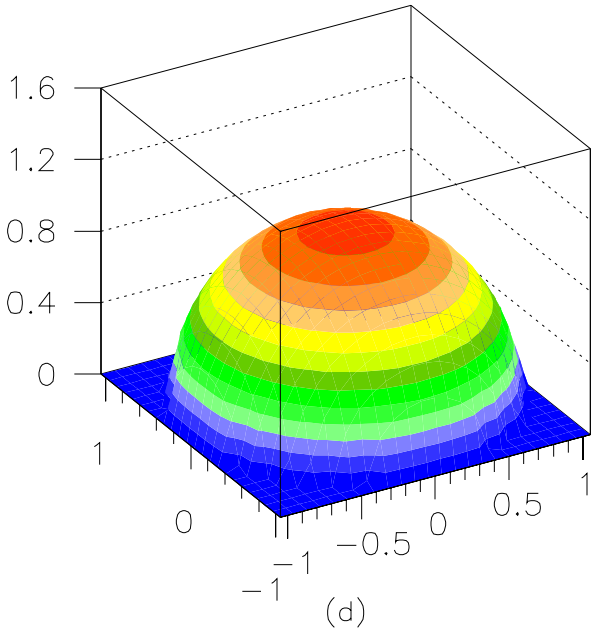
(a)



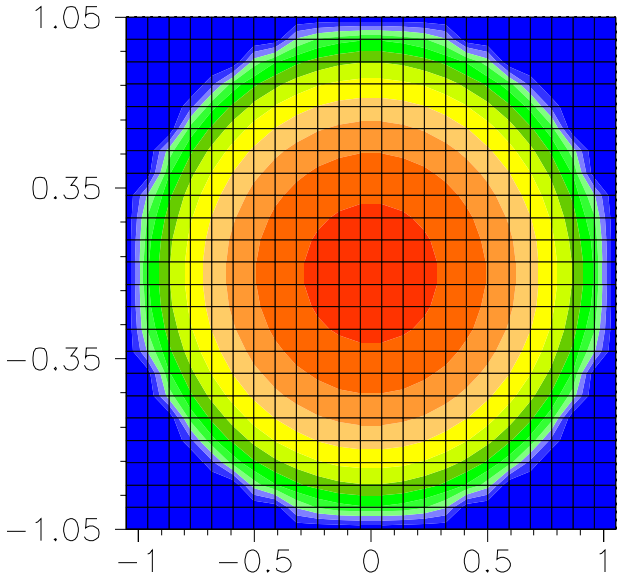
(b)



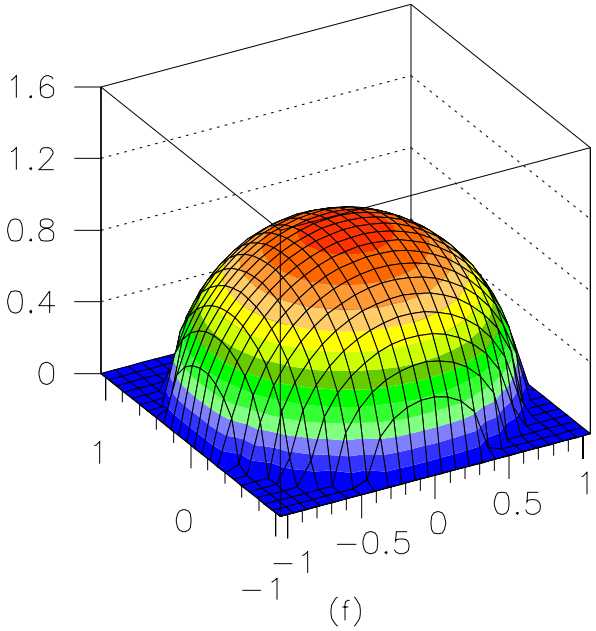
(c)



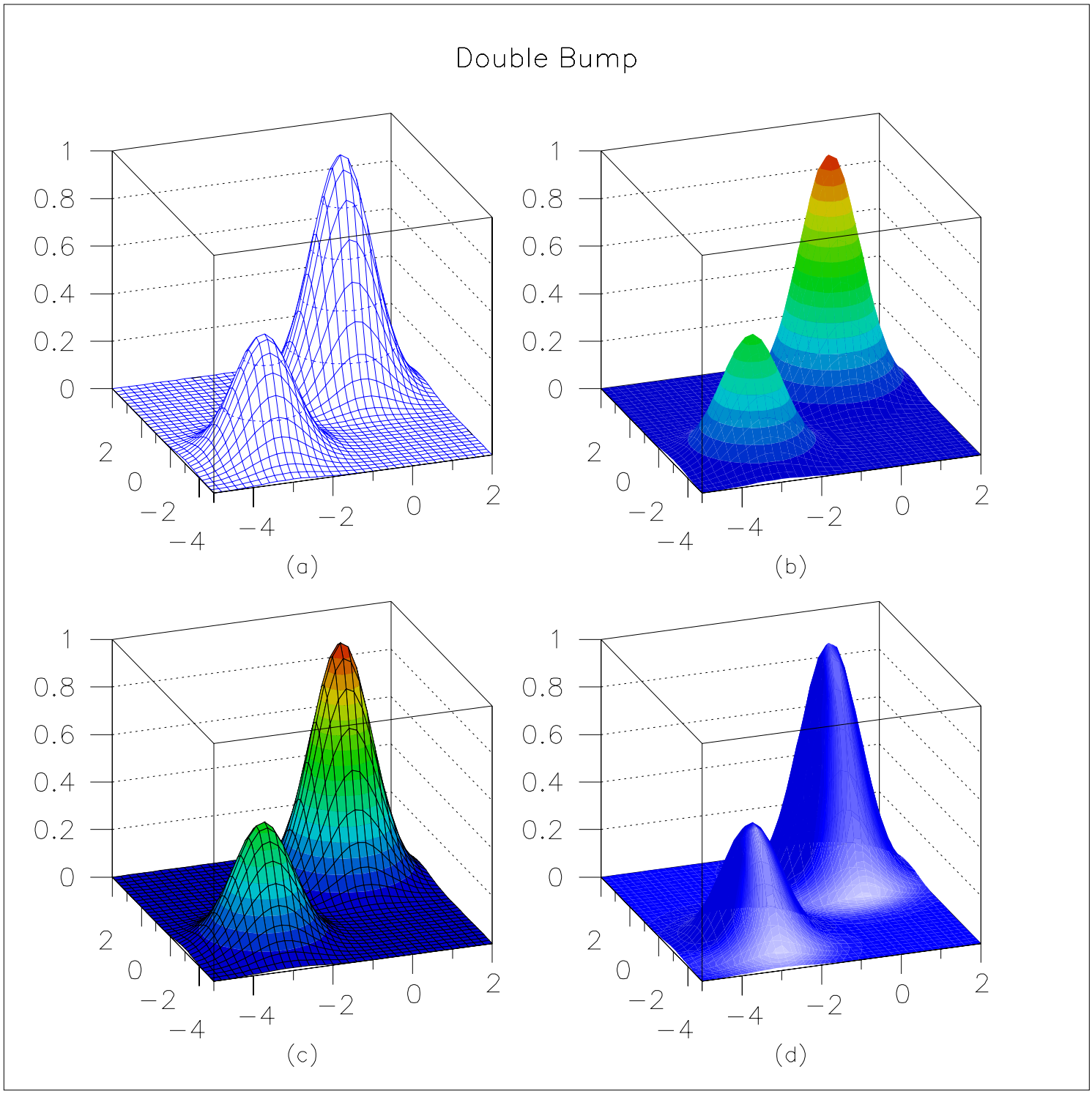
(d)



(e)



(f)


Double Bump
1
0.8
0.6
0.4
0.2
0
2
0
−2
−4
−4
−2
0
2
(a)
(b)
(c)
(d)