Welcome to the FAQ for comp.graphics.algorithms!

orourke@cs.smith.edu

Thanks to all who have contributed. Corrections and contributions (to orourke@cs.smith.edu) always welcome. Changed items this posting (|): 0.04, 0.07, 1.06, 3.01, 5.09, 7.02 New items this posting (+): none

Contents

1 0. General Information

1.1 : Charter of comp.graphics.algorithms

comp.graphics.algorithms is an unmoderated newsgroup intended as a forum for the discussion of the algorithms used in the process of generating computer graphics. These algorithms may be recently proposed in published journals or papers, old or previously known algorithms, or hacks used incidental to the process of computer graphics. The scope of these algorithms may range from an efficient way to multiply matrices, all the way to a global illumination method incorporating raytracing, radiosity, infinite spectrum modeling, and perhaps even mirrored balls and lime jello.

It is hoped that this group will serve as a forum for programmers and researchers to exchange ideas and ask questions on recent papers or current research related to computer graphics.

comp.graphics.algorithms is not:

- for requests for gifs, or other pictures - for requests for image translator or processing software; see alt.binaries.pictures* FAQ alt.binaries.pictures.utilities [now degenerated to pic postings] alt.graphics.pixutils (image format translation) comp.sources.misc (image viewing source code) sci.image.processing comp.graphics.apps.softimage fj.comp.image - for requests for compression software; for these try: alt.comp.compression comp.compression comp.compression.research

1.2 : Are the postings to comp.graphics.algorithms archived?

Archives may be found at: *http://www.faqs.org/*

1.3 : How can I get this FAQ?

The FAQ is posted on the 1st and 15th of every month. The easiest way to get it is to search back in your news reader for the most recent posting, with Subject: comp.graphics.algorithms Frequently Asked Questions It is posted to comp.graphics.algorithms, and cross-posted to news.answers and comp.answers.

If you can't find it on your newsreader, you can look at the latest HTML version at either of these two sites: *http://www.exaflop.org/docs/cgafaq* *http://www.faqs.org/* Both versions should be up-to-date and nicely converted. The maintainer also keeps a copy of the raw ASCII, accessible via *http://cs.smith.edu/* orourke/.

Finally, you can ftp the FAQ from several sites, including:

ftp://rtfm.mit.edu/pub/faqs/graphics/algorithms-faq *ftp://ftp.seas.gwu.edu/pub/rtfm/comp/graphics/algorithms/comp.graphics.algorithms_Frequently_Asked_Questions*

The (busy) rtfm.mit.edu site lists many alternative "mirror" sites. Also can reach the FAQ from *http://www.geom.umn.edu/software/cglist/*, which is worth visiting in its own right.

1.4 : What are some must-have books on graphics algorithms?

The keywords in brackets are used to refer to the books in later questions. They generally refer to the first author except where it is necessary to resolve ambiguity or in the case of the Gems.

Basic computer graphics, rendering algorithms, ——————————————————-

[Foley] Computer Graphics: Principles and Practice (2nd Ed.), J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, Addison-Wesley 1990, ISBN 0-201-12110-7; Computer Graphics: Principles and Practice, C version J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, Addison-Wesley ISBN: 0-201-84840-6, 1996, 1147 pp.

[Rogers:Procedural] Procedural Elements for Computer Graphics, Second Edition David F. Rogers, WCB/McGraw Hill 1998, ISBN 0-07-053548-5

[Rogers:Mathematical] Mathematical Elements for Computer Graphics 2nd Ed., David F. Rogers and J. Alan Adams, McGraw Hill 1990, ISBN 0-07-053530-2

[Watt:3D] _3D Computer Graphics, 2nd Edition_, Alan Watt, Addison-Wesley 1993, ISBN 0-201-63186-5

[Glassner:RayTracing] An Introduction to Ray Tracing, Andrew Glassner (ed.), Academic Press 1989, ISBN 0-12-286160-4

[Gems I] Graphics Gems, Andrew Glassner (ed.), Academic Press 1990, ISBN 0-12-286165-5

[Gems II] Graphics Gems II, James Arvo (ed.), Academic Press 1991, ISBN 0-12-64480-0

[Gems III] Graphics Gems III, David Kirk (ed.), Academic Press 1992, ISBN 0-12-409670-0 (with IBM disk) or 0-12-409671-9 (with Mac disk) See also "AP Professional Graphics CD-ROM Library," Academic Press, ISBN 0-12-059756-X, which contains Gems I-III.

[Gems IV] Graphics Gems IV, Paul S. Heckbert (ed.), Academic Press 1994, ISBN 0-12-336155-9 (with IBM disk) or 0-12-336156-7 (with Mac disk)

[Gems V] Graphic Gems V, Alan W. Paeth (ed.), Academic Press 1995, ISBN 0-12-543455-3 (with IBM disk)

[Watt:Animation] Advanced Animation and Rendering Techniques, Alan Watt, Mark Watt, Addison-Wesley 1992, ISBN 0-201-54412-1

[Bartels] An Introduction to Splines for Use in Computer Graphics and Geometric Modeling, Richard H. Bartels, John C. Beatty, Brian A. Barsky, 1987, ISBN 0-934613-27-3

[Farin] Curves and Surfaces for Computer Aided Geometric Design: | A Practical Guide, 4th Edition, Gerald E. Farin, Academic Press | 1996. ISBN 0122490541.

[Prusinkiewicz] The Algorithmic Beauty of Plants, Przemyslaw W. Prusinkiewicz, Aristid Lindenmayer, Springer-Verlag, 1990, ISBN 0-387-97297-8, ISBN 3-540-97297-8

[Oliver] Tricks of the Graphics Gurus, Dick Oliver, et al. (2) 3.5 PC disks included, $39.95 SAMS Publishing

[Hearn] Introduction to computer graphics, Hearn & Baker

[Cohen] Radiosity and Realistic Imange Sythesis, Michael F. Cohen, John R. Wallace, Academic Press Professional 1993, ISBN 0-12-178270-0 [apparently now out of print]

[Ashdown] Radiosity: A Programmer's Perspective Ian Ashdown, John Wiley & Sons 1994, ISBN 0-471-30444-1, 498 pp.

[Sillion] Radiosity & Global Illumination Francois X. Sillion snd Claude Puech, Morgan Kaufmann 1994, ISBN 1-55860-277-1, 252 pp.

[Ebert] Texturing and Modeling - A Procedural Approach (2nd Ed.) David S. Ebert (ed.), F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley, Academic Press 1998, ISBN 0-12-228730-4, Includes CD-ROM.

[Schroeder] Visualization Toolkit, 2nd Edition, The: An Object-Oriented Approach to 3-D Graphics (Bk/CD) (Professional Description) William J. Schroeder, Kenneth Martin, and Bill Lorensen, Prentice-Hall 1998, ISBN: 0-13-954694-4 See Subject 0.07 for source.

[Anderson] PC Graphics Unleashed Scott Anderson. SAMS Publishing, ISBN 0-672-30570-4

[Ammeraal] Computer Graphics for Java Programmers, Leen Ammeraal, John Wiley 1998, ISBN 0-471-98142-7. Additional information at *http://home.wxs.nl/* ammeraal/ .

For image processing, ——————

[Barnsley] Fractal Image Compression, Michael F. Barnsley and Lyman P. Hurd, AK Peters, Ltd, 1993 ISBN 1-56881-000-8

[Jain] Fundamentals of Image Processing, Anil K. Jain, Prentice-Hall 1989, ISBN 0-13-336165-9

[Castleman] Digital Image Processing, Kenneth R. Castleman, Prentice-Hall 1996, ISBN(Cloth): 0-13-211467-4 (Description and errata at: "*http://www.phoenix.net/* castlman")

[Pratt] Digital Image Processing, Second Edition, William K. Pratt, Wiley-Interscience 1991, ISBN 0-471-85766-1

[Gonzalez] Digital Image Processing (3rd Ed.), Rafael C. Gonzalez, Paul Wintz, Addison-Wesley 1992, ISBN 0-201-50803-6

[Russ] The Image Processing Handbook (3rd Ed.), John C. Russ, CRC Press and IEEE Press 1998, ISBN 0-8493-2532-3 [Russ & Russ] The Image Processing Tool Kit v. 3.0 Chris Russ and John Russ, Reindeer Games Inc. 1999, ISBN 1-928808-00-X

[Wolberg] Digital Image Warping, George Wolberg, IEEE Computer Society Press Monograph 1990, ISBN 0-8186-8944-7

Computational geometry —————-

[Bowyer] A Programmer's Geometry, Adrian Bowyer, John Woodwark, Butterworths 1983, ISBN 0-408-01242-0 Pbk

[O'Rourke (C)] Computational Geometry in C (2nd Ed.) Joseph O'Rourke, Cambridge University Press 1998, ISBN 0-521-64010-5 Pbk, ISBN 0-521-64976-5 Hbk Additional information and code at *http://cs.smith.edu/* orourke/ .

[O'Rourke (A)] Art Gallery Theorems and Algorithms Joseph O'Rourke, Oxford University Press 1987, ISBN 0-19-503965-3.

[Goodman & O'Rourke] Handbook of Discrete and Computational Geometry J. E. Goodman and J. O'Rourke, editors. CRC Press LLC, July 1997. ISBN:0-8493-8524-5 Additional information at *http://cs.smith.edu/* orourke/ .

[Samet:Application] Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Hanan Samet, Addison-Wesley, Reading, MA, 1990. ISBN 0-201-50300-0.

[Samet:Design & Analysis] The Design and Analysis of Spatial Data Structures, Hanan Samet, Addison-Wesley, Reading, MA, 1990. ISBN 0-201-50255-0.

[Mortenson] Geometric Modeling, Michael E. Mortenson, Wiley 1985, ISBN 0-471-88279-8

[Preparata] Computational Geometry: An Introduction, Franco P. Preparata, Michael Ian Shamos, Springer-Verlag 1985, ISBN 0-387-96131-3

[Okabe] Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, A. Okabe and B. Boots and K. Sugihara, John Wiley, Chichester, England, 1992.

[Overmars] Computational Geometry: Algorithms and Applications M. de Berg and M. van Kreveld and M. Overmars and O. Schwarzkopf Springer-Verlag, Berlin, 1997.

[Stolfi] Oriented Projective Geometry: A Framework for Geometric Computations Academic Press, 1991.

[Hodge] Methods of Algebraic Geometry, Volume 1 W.V.D. Hodge and D. Pedoe, Cambridge, 1994. ISBN 0-521-469007-4 Paperback

Algorithms books with chapters on computational geometry ——————————————

[Cormen et al.] Introduction to Algorithms, T. H. Cormen, C. E. Leiserson, R. L. Rivest, The MIT Press, McGraw-Hill, 1990.

[Mehlhorn] Data Structures and Algorithms, K. Mehlhorn, Springer-Verlag, 1984.

[Sedgewick] R. Sedgewick, Algorithms, Addison-Wesley, 1988.

Solid Modelling ————

[Mantyla] Introduction to Solid Modeling Martti Mantyla, Computer Science Press 1988, ISBN 07167-8015-1

1.5 : Are there any online references?

The computational geometry community maintains its own bibliography of publications in or closely related to that subject. Every four months, additions and corrections are solicited from users, after which the database is updated and released anew. As of 31 Mar 1998, it contained 11,275 bib-tex entries. See Jeff Erickson's page on "Computational Geometry Bibliographies": *http://www.cs.duke.edu/* jeffe/compgeom/biblios.html#geombib The bibliography can be retrieved from:

ftp://ftp.cs.usask.ca/pub/geometry/geombib.tar.gz - bibliography proper *ftp://ftp.cs.usask.ca/pub/geometry/o-*cgc19.ps.gz - overview published in '93 in SIGACT News and the Internat. J. Comput. Geom. Appl. *ftp://ftp.cs.usask.ca/pub/geometry/ftp*-hints - detailed retrieval info

Universitat Politecnica de Catalunya maintains a search engine at: *http://www*-ma2.upc.es/ geomc/geombibe.html

The ACM SIGGRAPH Online Bibliography Project, by Stephen Spencer (biblio@siggraph.org). The database is available for anonymous FTP from the *ftp://siggraph.org/publications/bibliography* directory. Please download and examine the file READ_ME in that directory for more specific information concerning the database.

'netlib' is a useful source for algorithms, member inquiries for SIAM, and bibliographic searches. For information, send mail to netlib@ornl.gov, with "send index" in the body of the mail message.

You can also find free sources for numerical computation in C via *ftp://usc.edu/pub/C*-numanal. In particular, grab numcomp-free-c.gz in that directory.

Check out Nick Fotis's computer graphics resources FAQ – it's packed with pointers to all sorts of great computer graphics stuff. This FAQ is posted biweekly to comp.graphics.

This WWW page contains links to a large number of computer graphic related pages: *http://www.dataspace.com*:84/vlib/comp-graphics.html

There's a Computer Science Bibliography Server at: *http://glimpse.cs.arizona.edu*:1994/bib/ with Computer Graphics, Vision and Radiosity sections

A comprehensive bibliography of color quantization papers and articles (CQUANT97) is available at *http://www.ledalite.com/library-*/cgis.htm.

Modelling physically based systems for animation: *http://www.cc.gatech.edu/gvu/animation/Animation.html*

The University of Manchester NURBS Library: *ftp://unix.hensa.ac.uk/pub/misc/unix/nurbs/*

For an implementation of Seidel's algorithm for fast trapezoidation and triangulation of polygons. You can get the code from: *ftp://ftp.cs.unc.edu/pub/users/narkhede/triangulation.tar.gz*

Ray tracing bibliography: *ftp://ftp.eye.com/pub/graphics/papers/rtbib95.tar.Z* *ftp://ftp.eye.com/pub/graphics/papers/rtbib95.zip*

Quaternions and other comp sci curiosities: *ftp://ftp.netcom.com/pub/hb/hbaker/hakmem/hakmem.html*

Directory of Computational Geometry Software, collected by Nina Amenta (nina@cs.utexas.edu) Nina Amenta is maintaining a WWW directory to computational geometry software. The directory lives at The Geometry Center. It has pointers to lots of convex hull and voronoi diagram programs, triangulations, collision detection, polygon intersection, smallest enclosing ball of a point set and other stuff. *http://www.geom.umn.edu/software/cglist/*

A compact reference for real-time 3d computer graphics programming: *http://www.math.mcgill.ca/* loisel/

RADBIB99 is a comprehensive bibliography of radiosity and related global illumination papers, theses, articles, and books. It currently includes 1,593 references. This bibliography is available in BibTex format as RADBIB99.BIB (with a release date of June 1, 1999) from: *http://www.helios32.com/radbib99.bib*

The "Electronic Visualization Library" (EVlib) is a domain- secific digital library for Scientific Visualization and Computer Graphics: *http://visinfo.zib.de/*

1.6 : Are there other graphics related FAQs?

BSP Tree FAQ by Bretton Wade *http://reality.sgi.com/bspfaq/*

Gamma and Color FAQs by Charles A. Poynton has *ftp://ftp.inforamp.net/pub/users/poynton/doc/colour/* *http://www.inforamp.net/* poynton/

The documents are mirrored in Darmstadt, Germany at *ftp://ftp.igd.fhg.de/pub/doc/colour/*

1.7 : Where is all the source?

Graphics Gems source code. *http://www.acm.org/tog/GraphicsGems/* This site is now the offical distribution site for Graphics Gems code.

Master list of Computational Geometry software: *http://www.geom.umn.edu/software/cglist* Described in [Goodman & O'Rourke], Chap. 52.

Jeff Erikson's software list: *http://compgeom.cs.uiuc.edu/* jeffe/compgeom/compgeom.html

General 'stuff' *ftp://wuarchive.wustl.edu/graphics/graphics*

There are a number of interesting items in *http://graphics.lcs.mit.edu/* seth including: - Code for 2D Voronoi, Delaunay, and Convex hull - Mike Hoymeyer's implementation of Raimund Seidel's O(d! n) time linear programming algorithm for n constraints in d dimensions - geometric models of UC Berkeley's new computer science building

Sources to "Computational Geometry in C", by J. O'Rourke can be found at *ftp://cs.smith.edu/pub/compgeom*

Greg Ferrar has uploaded his heavily commented C++ 3D rendering library at *ftp://ftp.math.ohio-*state.edu/pub/users/gregt

TAGL is a portable and extensible library that provides a subset of Open-GL functionalities. *ftp://sunsite.unc.edu/pub/packages/programming/graphics/tagl21.tgz*

Try *ftp://x2ftp.oulu.fi* for /pub/msdos/programming/docs/graphpro.lzh by Michael Abrash. His XSharp package has an implementation of Xiaoulin Wu's anti-aliasing algorithm (in C).

Example sources for BSP tree algorithms can be found at | *http://reality.sgi.com/bspfaq/*, item 24.

Mel Slater (mel@dcs.qmw.ac.uk) also made some implementations of BSP trees and shadows for static scenes using shadow volumes code available *http://www.dcs.qmw.ac.uk/* mel/BSP.html *ftp://ftp.dcs.qmw.ac.uk/people/mel/BSP*

The Visualization Toolkit (A visualization textbook, C++ library and Tcl-based interpreter) (see [Schroeder]): *http://www.kitware.com/vtk.html*

WINGED.ZIP, a C++ implementation of Baumgart's winged-edge data structure: *http://www.ledalite.com/library-/*cgis.htm

CGAL, the Computational Geometry Algorithms Library, is written in C++ and is available at URL *http://www.cs.ruu.nl/CGAL/* . It consists of three parts. The first part is the kernel, which consists of constant size non-modifiable geometric primitive objects. The second part is a collection of basic geometric datastructures and algorithms, which are parameterized by traits classes that define the interface between the datastructure or algorithm, and the primitives they use. The third part consists of non-geometric support facilities.

A C++ NURBS library written by Lavoie Philippe. Version 2.1. Results may be exported as POV-Ray, RIB (renderman) or VRML files. It also offers wrappers to OpenGL: | *http://yukon.genie.uottawa.ca/* lavoie/software/nurbs/

Paul Bourke has code for several problems, including isosurface generation and Delauney triangulation, at: *http://www.mhri.edu.au/* pdb/geometry/ *http://www.mhri.edu.au/* pdb/modeling/

A nearly comprehensive list of available 3D engines (most with source code): *http://cg.cs.tu-*berlin.de/ ki/engines.html

See also 5.17: Where can I get the spline description of the famous teapot etc.?

Interactive Geometry Software called "Cinderella": *http://www.cinderella.de*

2 1. 2D Computations: Points, Segments, Circles, Etc.

2.1 : How do I rotate a 2D point?

In 2-D, the 2x2 matrix is very simple. If you want to rotate a column vector v by t degrees using matrix M, use

M = {{cos t, -sin t}, {sin t, cos t}} in M*v.

If you have a row vector, use the transpose of M (turn rows into columns and vice versa). If you want to combine rotations, in 2-D you can just add their angles, but in higher dimensions you must multiply their matrices.

2.2 : How do I find the distance from a point to a line?

Let the point be C (Cx,Cy) and the line be AB (Ax,Ay) to (Bx,By). Let P be the point of perpendicular projection of C on AB. The parameter r, which indicates P's position along AB, is computed by the dot product of AC and AB divided by the square of the length of AB:

```
(1)     AC dot AB
```

```
    r = ---------
            ||AB||^2

r has the following meaning:

    r=0      P = A
    r=1      P = B
    r<0      P is on the backward extension of AB
    r>1      P is on the forward extension of AB
    0<r<1    P is interior to AB

The length of a line segment in d dimensions, AB is computed by:

    L = sqrt( (Bx-Ax)^2 + (By-Ay)^2 + ... + (Bd-Ad)^2)
```

so in 2D:

L = sqrt((Bx-Ax)ˆ2 + (By-Ay)ˆ2)

and the dot product of two vectors in d dimensions, U dot V is computed:

D = (Ux * Vx) + (Uy * Vy) + ... + (Ud * Vd)

so in 2D:

D = (Ux * Vx) + (Uy * Vy)

So (1) expands to:

(Cx-Ax)(Bx-Ax) + (Cy-Ay)(By-Ay) r = ——————————————- Lˆ2

The point P can then be found:

Px = Ax + r(Bx-Ax) Py = Ay + r(By-Ay)

And the distance from A to P = r*L.

Use another parameter s to indicate the location along PC, with the following meaning: s<0 C is left of AB s>0 C is right of AB s=0 C is on AB

Compute s as follows:

(Ay-Cy)(Bx-Ax)-(Ax-Cx)(By-Ay) s = ————————————— Lˆ2

Then the distance from C to P = s*L.

2.3 : How do I find intersections of 2 2D line segments?

This problem can be extremely easy or extremely difficult depends on your applications. If all you want is the intersection point, the following should work:

Let A,B,C,D be 2-space position vectors. Then the directed line segments AB & CD are given by:

AB=A+r(B-A), r in [0,1] CD=C+s(D-C), s in [0,1]

If AB & CD intersect, then

A+r(B-A)=C+s(D-C), or

Ax+r(Bx-Ax)=Cx+s(Dx-Cx) Ay+r(By-Ay)=Cy+s(Dy-Cy) for some r,s in [0,1]

Solving the above for r and s yields

(Ay-Cy)(Dx-Cx)-(Ax-Cx)(Dy-Cy) r = ————————————— (eqn 1) (Bx-Ax)(Dy-Cy)-(By-Ay)(Dx-Cx)

(Ay-Cy)(Bx-Ax)-(Ax-Cx)(By-Ay) s = ————————————— (eqn 2) (Bx-Ax)(Dy-Cy)-(By-Ay)(Dx-Cx)

Let P be the position vector of the intersection point, then

P=A+r(B-A) or

Px=Ax+r(Bx-Ax) Py=Ay+r(By-Ay)

By examining the values of r & s, you can also determine some other limiting conditions:

If 0<=r<=1 & 0<=s<=1, intersection exists r<0 or r>1 or s<0 or s>1 line segments do not intersect

If the denominator in eqn 1 is zero, AB & CD are parallel If the numerator in eqn 1 is also zero, AB & CD are collinear.

If they are collinear, then the segments may be projected to the x- or y-axis, and overlap of the projected intervals checked.

If the intersection point of the 2 lines are needed (lines in this context mean infinite lines) regardless whether the two line segments intersect, then

---

```
        If r>1, P is located on extension of AB
        If r<0, P is located on extension of BA
        If s>1, P is located on extension of CD
        If s<0, P is located on extension of DC
```

Also note that the denominators of eqn 1 & 2 are identical.

References:

[O'Rourke (C)] pp. 249-51 [Gems III] pp. 199-202 "Faster Line Segment Intersection,"

2.4 : How do I generate a circle through three points?

Let the three given points be a, b, c. Use _0 and _1 to represent x and y coordinates. The coordinates of the center p=(p_0,p_1) of the circle determined by a, b, and c are:

A = b_0 - a_0; B = b_1 - a_1; C = c_0 - a_0; D = c_1 - a_1;

E = A*(a_0 + b_0) + B*(a_1 + b_1); F = C*(a_0 + c_0) + D*(a_1 + c_1);

G = 2.0*(A*(c_1 - b_1)-B*(c_0 - b_0));

p_0 = (D*E - B*F) / G; p_1 = (A*F - C*E) / G;

If G is zero then the three points are collinear and no finite-radius circle through them exists. Otherwise, the radius of the circle is:

$$r^2 = (a_0 - p_0)^2 + (a_1 - p_1)^2$$

Reference:

[O' Rourke (C)] p. 201. Simplified by Jim Ward.

2.5 : How can the smallest circle enclosing a set of points be found?

This circle is often called the minimum spanning circle. It can be computed in O(n log n) time for n points. The center lies on the furthest-point Voronoi diagram. Computing the diagram constrains the search for the center. Constructing the diagram can be accomplished by a 3D convex hull algorithm; for that connection, see, e.g., [O'Rourke (C), p.195ff]. For direct algorithms, see: S. Skyum, "A simple algorithm for computing the smallest enclosing circle" Inform. Process. Lett. 37 (1991) 121–125. J. Rokne, "An Easy Bounding Circle" [Gems II] pp.14–16.

2.6 : Where can I find graph layout algorithms?

See the paper by Eades and Tamassia, "Algorithms for Drawing Graphs: An Annotated Bibliography," Tech Rep CS-89-09, Dept. of CS, Brown Univ, Feb. 1989.

A revised version of the annotated bibliography on graph drawing algorithms by Giuseppe Di Battista, Peter Eades, and Roberto Tamassia is now available from

ftp://wilma.cs.brown.edu/pub/papers/compgeo/gdbiblio.tex.gz and *ftp://wilma.cs.brown.edu/pub/papers/compgeo/gdbiblio.ps.gz*

Commercial software includes the Graph Layout Toolkit from Tom Sawyer | Software *http://www.tomsawyer.com* and Northwoods Software's GO++ | *http://www.nwoods.com/go/* .

3 2. 2D Polygon Computations

3.1 : How do I find the area of a polygon?

The signed area can be computed in linear time by a simple sum. The key formula is this:

If the coordinates of vertex v_i are x_i and y_i, twice the signed area of a polygon is given by

$$2 A(P) = \sum_{i=0}^{n-1} (x_i y_{i+1} - y_i x_{i+1}).$$

Here n is the number of vertices of the polygon. A rearrangement of terms in this equation can save multiplications and operate on coordinate differences, and so may be both faster and more accurate:

$$2 A(P) = \sum_{i=0}^{n-1} ((x_i + x_{i+1}) (y_{i+1} - y_i)).$$

References: [O' Rourke (C)] Thm. 1.3.3, p. 21; [Gems II] pp. 5-6: "The Area of a Simple Polygon", Jon Rokne.

To find the area of a planar polygon not in the x-y plane, use:

$$2 A(P) = \mathrm{abs}(N \cdot (\sum_{i=0}^{n-1} (v_i \times v_{i+1})))$$

where N is a unit vector normal to the plane. The '.' represents the dot product operator, the 'x' represents the cross product operator, and abs() is the absolute value function.

[Gems II] pp. 170-171: "Area of Planar Polygons and Volume of Polyhedra", Ronald N. Goldman.

3.2 : How can the centroid of a polygon be computed?

The centroid (a.k.a. the center of mass, or center of gravity) of a polygon can be computed as the weighted sum of the centroids of a partition of the polygon into triangles. The centroid of a triangle is simply the average of its three vertices, i.e., it has coordinates (x1 + x2 + x3)/3 and (y1 + y2 + y3)/3. This suggests first triangulating the polygon, then forming a sum of the centroids of each triangle, weighted by the area of each triangle, the whole sum normalized by the total polygon area. This indeed works, but there is a simpler method: the triangulation need not be a partition, but rather can use positively and negatively oriented triangles (with positive and negative areas), as is used when computing the area of a polygon. This leads to a very simple algorithm for computing the centroid, based on a sum of triangle centroids weighted with their signed area. The triangles can be taken to be those formed by any fixed point, e.g., the vertex v0 of the polygon, and the two endpoints of consecutive edges of the polygon: (v1,v2), (v2,v3), etc. The area of a triangle with vertices a, b, c is half of this expression:

```
(b[X] - a[X]) * (c[Y] - a[Y]) -
(c[X] - a[X]) * (b[Y] - a[Y]);
```

Code available at *ftp://cs.smith.edu/pub/code/centroid.c* (3K). Reference: [Gems IV] pp.3-6; also includes code.

3.3 : How do I find if a point lies within a polygon?

The definitive reference is "Point in Polyon Strategies" by Eric Haines [Gems IV] pp. 24-46. The code in the Sedgewick book Algorithms (2nd Edition, p.354) is incorrect.

The essence of the ray-crossing method is as follows. Think of standing inside a field with a fence representing the polygon. Then walk north. If you have to jump the fence you know you are now outside the poly. If you have to cross again you know you are now inside again; i.e., if you were inside the field to start with, the total number of fence jumps you would make will be odd, whereas if you were ouside the jumps will be even.

The code below is from Wm. Randolph Franklin *wrf@ecse.rpi.edu* with some minor modifications for speed. It returns 1 for strictly interior points, 0 for strictly exterior, and 0 or 1 for points on the boundary. The boundary behavior is complex but determined; in particular, for a partition of a region into polygons, each point is "in" exactly one polygon. See the references below for more detail.

```
int pnpoly(int npol, float *xp, float *yp, float x, float y)
{
  int i, j, c = 0;
  for (i = 0, j = npol-1; i < npol; j = i++) {
    if ((((yp[i]<=y) && (y<yp[j])) ||
         ((yp[j]<=y) && (y<yp[i]))) &&
        (x < (xp[j] - xp[i]) * (y - yp[i]) / (yp[j] - yp[i]) + xp[i]))
```

c = !c; } return c; }

The code may be further accelerated, at some loss in clarity, by avoiding the central computation when the inequality can be deduced, and by replacing the division by a multiplication for those processors with slow divides. For code that distinguishes strictly interior points from those on the boundary, see [O'Rourke (C)] pp. 239-245.

References: [Gems IV] pp. 24-46 [O'Rourke (C)] [Glassner:RayTracing]

3.4 : How do I find the intersection of two convex polygons?

Unlike intersections of general polygons, which might produce a quadratic number of pieces, intersection of convex polygons of n and m vertices always produces a polygon of at most (n+m) vertices. There are a variety of algorithms whose time complexity is proportional to this size, i.e., linear. The first, due to Shamos and Hoey, is perhaps the easiest to understand. Let the two polygons be P and Q. Draw a horizontal line through every vertex of each. This partitions each into trapezoids (with at most two triangles, one at the top and one at the bottom). Now scan down the two polygons in concert, one trapezoid at a time, and intersect the trapezoids if they overlap vertically. A more difficult-to-describe algorithm is in [O'Rourke (C)], pp. 242-252. This walks around the boundaries of P and Q in concert, intersecting edges. An implementation is included in [O'Rourke (C)].

3.5 : How do I do a hidden surface test (backface culling) with 2d points?

c = (x1-x2)*(y3-y2)-(y1-y2)*(x3-x2)

x1,y1, x2,y2, x3,y3 = the first three points of the polygon.

If c is positive the polygon is visible. If c is negative the polygon is invisible (or the other way).

3.6 : How do I find a single point inside a simple polygon?

Given a simple polygon, find some point inside it. Here is a method based on the proof that there exists an internal diagonal, in [O'Rourke (C), 13-14]. The idea is that the midpoint of a diagonal is interior to the polygon.

1. Identify a convex vertex v; let its adjacent vertices be a and b. 2. For each other vertex q do: 2a. If q is inside avb, compute distance to v (orthogonal to ab). 2b. Save point q if distance is a new min. 3. If no point is inside, return midpoint of ab, or centroid of avb. 4. Else if some point inside, qv is internal: return its midpoint.

Code for finding a diagonal is in [O'Rourke (C), 35-39], and is part of many other software packages. See Subject 0.07: Where is all the source?

3.7 : How do I find the orientation of a simple polygon?

Compute the signed area (Subject 2.01). The orientation is counter-clockwise if this area is positive.

A slightly faster method is based on the observation that it isn't necessary to compute the area. Find the lowest vertex (or, if there is more than one vertex with the same lowest coordinate, the rightmost of those vertices) and then take the cross product of the edges fore and aft of it. Both methods are O(n) for n vertices, but it does seem a waste to add up the total area when a single cross product (of just the right edges) suffices. Code for this is available at *ftp://cs.smith.edu/pub/code/polyorient.C* (2K).

The reason that the lowest, rightmost (or any other such extreme) point works is that the internal angle at this vertex is necessarily convex, strictly less than pi (even if there are several equally-lowest points).

4 3. 2D Image/Pixel Computations

4.1 : How do I rotate a bitmap?

The easiest way, according to the comp.graphics faq, is to take the rotation transformation and invert it. Then you just iterate over the destination image, apply this inverse transformation and find which source pixel to copy there.

A much nicer way comes from the observation that the rotation matrix:

R(T) = { { cos(T), -sin(T) }, { sin(T), cos(T) } }

is formed my multiplying three matrices, namely:

R(T) = M1(T) * M2(T) * M3(T)

where

```
M1(T) = { { 1, -tan(T/2) },
          { 0, 1         } }
M2(T) = { { 1,      0    },
          { sin(T), 1    } }
M3(T) = { { 1, -tan(T/2) },
          { 0,         1 } }
```

Each transformation can be performed in a separate pass, and because these transformations are either row-preserving or column-preserving, anti-aliasing is quite easy.

| Another fast approach is to perform first a column-preserving roation, | and then a row-preserving rotation. For an image W pixels wide and | H pixels high, this requires W+H BitBlt operations in comparison to | the brute-force rotation, which uses W*H SetPixel operations (and a | lot of multiplying).

Reference:

Paeth, A. W., "A Fast Algorithm for General Raster Rotation", Proceedings Graphics Interface '89, Canadian Information Processing Society, 1986, 77-81 [Note - e-mail copies of this paper are no longer available]

[Gems I]

4.2 : How do I display a 24 bit image in 8 bits?

[Gems I] pp. 287-293, "A Simple Method for Color Quantization: Octree Quantization"

B. Kurz. Optimal Color Quantization for Color Displays. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1983, pp. 217-224.

[Gems II] pp. 116-125, "Efficient Inverse Color Map Computation"

This describes an efficient technique to map actual colors to a reduced color map, selected by some other technique described in the other papers.

[Gems II] pp. 126-133, "Efficient Statistical Computations for Optimal Color Quantization"

Xiaolin Wu. Color Quantization by Dynamic Programming and Principal Analysis. ACM Transactions on Graphics, Vol. 11, No. 4, October 1992, pp 348-372.

4.3 : How do I fill the area of an arbitrary shape?

"A Fast Algorithm for the Restoration of Images Based on Chain Codes Description and Its Applications", L.W. Chang & K.L. Leu, Computer Vision, Graphics, and Image Processing, vol.50, pp296-307 (1990)

"An Introductory Course in Computer Graphics" by Richard Kingslake, (2nd edition) published by Chartwell-Bratt ISBN 0-86238-284-X

[Gems I] [Foley] [Hearn]

4.4 : How do I find the 'edges' in a bitmap?

A simple method is to put the bitmap through the filter:

-1 -1 -1 -1 8 -1 -1 -1 -1

This will highlight changes in contrast. Then any part of the picture where the absolute filtered value is higher than some threshold is an "edge".

A more appropriate edge detector for noisy images is described by Van Vliet et al. "A nonlinear Laplace operator as edge detector in noisy images", in Computer Vision, Graphics, and image processing 45, pp. 167-195, 1989.

4.5 : How do I enlarge/sharpen/fuzz a bitmap?

Sharpening of bitmaps can be done by the following algorithm:

I_enh(x,y) = I_fuz(x,y)-k*Laplace(I_fuz(x,y))

or in words: An image can be sharpened by subtracting a positive fraction k of the Laplace from the fuzzy image.

The Laplace is the kernal:

```
1   1   1
1  -8   1
1   1   1
```

The following library implements Fast Gaussian Blurs:

MAGIC: An Object-Oriented Library for Image Analysis by David Eberly

The library source code and the documentation (in Latex) are at *http://www.magic*-software.com/ The code compiles on Unix systems using g++ and on PCs using Microsoft Windows 3.1 and Borland C++. The fast Gaussian blurring is based on a finite difference method for solving s u_s = s^2 \nabla^2 u where s is the standard deviation of the Gaussian (t = s^2/2). It takes advantage of geometrically increasing steps in s (rather than linearly increasing steps in t), thus getting to a larger "time" rapidly, but still retaining stability. Section 4.5 of the documentation contains the algorithm description and implementation.

A bitmap is a sampled image, a special case of a digital signal, and suffers from two limitations common to all digital signals. First, it cannot provide details at fine enough spacing to exactly reproduce every continuous image, nor even more detailed sampled images. And second, each sample approximates the infinitely fine variability of ideal values with a discrete set of ranges encoded in a small number of bits—sometimes just one bit per pixel. Many times bitmaps have another limitation imposed: The values canot be negative. The resolution limitation is perhaps most important.

The ideal way to enlarge a bitmap is to work from the original continuous image, magnifying and resampling it. The standard way to do it in practice is to (conceptually) reconstruct a continuous image from the bitmap, and magnify and resample that instead. This will not give the same results, since details of the original have already been lost, but it is the best approach possible given an already sampled image. More details are provided below.

Both sharpening and fuzzing are examples of filtering. Even more specifically, they can be both be accomplished with filters which are linear and shift invariant. A crude way to sharpen along a row (or column) is to set output pixel B[n] to the difference of input pixels, A[n]-A[n-1]. A similarly crude way to fuzz is to set B[n] to the average of input pixels, 1/2*A[n]+1/2*A[n-1]. In each case the output is a weighted sum of input pixels, a "convolution". One important characteristic of such filters is that a sinusoid going in produces a sinusoid coming out, one of the same frequency. Thus the Fourier transform, which decomposes a signal into sinusoids of various frequencies, is the key to analysis of these filters. The simplest (and most efficient) way to handle the two dimensions of images is to operate on first the rows then the columns (or vice versa). Fourier transforms and many filters allow this separation.

A filter is linear if it satisfies two simple relations between the input and output: scaling the input by a factor scales the output by the same factor, and the sum of two inputs gives the sum of the two outputs. A filter is shift invariant if shifting the input up, down, left, or right merely shifts the output the same way. When a filter is both linear and shift invariant, it can be implemented as a convolution, a weighted sum. If you find the output of the filter when the input is a single pixel with value one in a sea of zeros, you will know all the weights. This output is the impulse response of the filter. The Fourier transform of the impulse response gives the frequency response of the filter. The pattern of weights read off from the impulse response gives the filter kernel, which will usually be displayed (for image filters) as a 2D stencil array, and it is almost always symmetric around the center. For example, the following filter, approximating a Laplacian (and used for detecting edges), is centered on the negative value.

```
        1/6     4/6     1/6
        4/6   -20/6     4/6
        1/6     4/6     1/6
```

The symmetry allows a streamlined implementation. Suppose the input image is in A, and the output is to go into B. Then compute B[i][j] = (A[i-1][j-1]+A[i-1][j+1]+A[i+1][j-1]+A[i+1][j+1] +4.0*(A[i-1][j]+A[i][j-1]+A[i][j+1]+A[i+1][j]) -20.0*A[i][j])/6.0

Ideal blurring is uniform in all directions, in other words it has circular symmetry. Gaussian blurs are popular, but the obvious code is slow for wide blurs. A cheap alternative is the following filter (written for rows, but then applied to the columns as well). B[i][j] = ((A[i][j]*2+A[i][j-1]+A[i][j+1])*4 +A[i][j-1]+A[i][j+1]-A[i][j-3]-A[i][j+3])/16 For sharpening, subtract the results from the original image, which is equivalent to using the following. B[i][j] = ((A[i][j]*2-A[i][j-1]-A[i][j+1])*4 -A[i][j-1]-A[i][j+1]+A[i][j-3]+A[i][j+3])/16 Credit for this filter goes to Ken Turkowski and Steve Gabriel.

Reconstruction is impossible without some assumptions, and because of the importance of sinusoids in filtering it is traditional to assume the continuous image is made of sinusoids mixed together. That makes more sense for sounds, where signal processing began, than it does for images, especially computer images of character shapes, sharp surface features, and halftoned shading. As pointed out above, often image values cannot be negative, unlike sinusoids. Also, real world images contain noise. The best noise suppressors (and edge detectors) are, ironically, nonlinear filters.

The simplest way to double the size of an image is to use each of the original pixels twice in its row and in its column. For much better results, try this instead. Put zeros between the original pixels, then use the blurring filter given a moment ago. But you might want to divide by 8 instead of 16 (since the zeros will dim the image otherwise). To instead shrink the image by half (in both vertical and horizontal), first apply the filter (dividing by 16), then throw away every other pixel. Notice that there are obvious optimizations involving arithmetic with powers of two, zeros which are in known locations, and pixels which will be discarded.

4.6 : How do I map a texture on to a shape?

Paul S. Heckbert, "Survey of Texture Mapping", IEEE Computer Graphics and Applications V6, #11, Nov. 1986, pp 56-67 revised from Graphics Interface '86 version

Eric A. Bier and Kenneth R. Sloan, Jr., "Two-Part Texture Mappings", IEEE Computer Graphics and Applications V6 #9, Sept. 1986, pp 40-53 (projection parameterizations)

4.7 : How do I detect a 'corner' in a collection of points?

[Currently empty entry.]

4.8 : Where do I get source to display (raster font format)?

ftp://oak.oakland.edu/SimTel/msdos/graphics See also James Murray's graphics file formats FAQ: *http://www.ora.com/centers/gff/gff*-faq/index.htm

4.9 : What is morphing/how is it done?

See [Anderson], Chapter 3, page 59 - 90.

4.10 : How do I quickly draw a filled triangle?

The easiest way is to render each line separately into an edge buffer. This buffer is a structure which looks like this (in C):

struct { int xmin, xmax; } edgebuffer[YDIM];

There is one entry for each scan line on the screen, and each entry is to be interpreted as a horizontal line to be drawn from xmin to xmax.

Since most people who ask this question are trying to write fast games on the PC, I'll tell you where to find code. Look at:

ftp::/ftp.uwp.edu/pub/msdos/demos/programming/source ftp::/ftp.luth.se/pub/msdos/demos (Sweden) ftp::/NCTUCCCA.edu.tw:/PC/uwp/demos *http://www.wit.com*:/mirrors/uwp/pub/msdos/demos ftp::/ftp.cdrom.com:/demos

4.11 : 3D Noise functions and turbulence in Solid texturing.

See: *ftp://gondwana.ecr.mu.oz.au/pub/siggraph92_ C23.shar.gz* *ftp://ftp.cis.ohio-*state.edu/pub/siggraph92/siggraph92_C23.shar

In it there are implementations of Perlin's noise and turbulence functions, (By the man himself) as well as Lewis' sparse convolution noise function (by D. Peachey) There is also some of other stuff in there (Musgrave's Earth texture functions, and some stuff on animating gases by Ebert).

SPD (Standard Procedural Databases) package: *ftp://avalon.chinalake.navy.mil/utils/SPD/SPD33f4.tar.Z* *ftp://avalon.chinalake.navy.mil/utils/SPD/spd33f4.zip.* Now moved to *http://www.viewpoint.com/*

References:

[Ebert] Noise, Hypertexture, Antialiasing and Gesture, (Ken Perlin) in Chapter 6, (p.193-), The disk accompanying the book is available from *ftp://archive.cs.umbc.edu/pub/texture.*

For more info on this text/code see: *http://www.cs.umbc.edu/* ebert/book/book.html

For examples from a current course based on this book, see: *http://www.seas.gwu.edu/graphics/ProcTexCourse/*

[Watt:Animation] Three-dimensional Nocie, Chapter 7.2.1 Simulating turbulance, Chapter 7.2.2

4.12 : How do I generate realistic sythetic textures?

For fractal mountains, trees and sea-shells:

SPD (Standard Procedural Databases) package: *ftp://avalon.chinalake.navy.mil/utils/SPD/SPD33f4.tar.Z* *ftp://avalon.chinalake.navy.mil/utils/SPD/spd33f4.zip.* Now moved to *http://www.viewpoint.com/*

Reaction-Diffusion Algorithms: For an illustartion of the parameter space of a reaction diffusion system, check out the Xmorphia page at *http://www.ccsf.caltech.edu/ismap/image.html*

References:

[Ebert] Entire book devoted to this subject, with RenderMan(TM) and C code.

[Watt:Animation] Procedural texture mapping and modelling, Chapter 7

"Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion" Greg Turk, Computer Graphics, Vol. 25, No. 4, pp. 289-298 July 1991 (SIGGRAPH '91) *http://www.cs.unc.edu*:80/ turk/reaction_diffusion/reaction_diffusion.html

A list of procedural texture synthesis related web pages *http://www.threedgraphics.com/pixelloom/tex_synth.html*

4.13 : How do I convert between color models (RGB, HLS, CMYK, CIE etc)?

References: [Watt:3D] pp. 313-354 [Foley] pp. 563-603

4.14 : How is "GIF" pronounced?

"GIF" is an acronymn for "Graphics Interchange Format." Despite the hard "G" in "Graphics," GIF is pronounced "JIF." Although we don't have a direct quote from the official CompuServe specification released June 1987, here is a quote from related CompuServe documentation, for CompuShow, a DOS-based image viewer used shortly thereafter: "The GIF (Graphics Interchange Format), pronounced "JIF", was designed by CompuServe ..." We also have a report that the principal author of the GIF spec, Bob Berry, pronounced it "JIF." Anyone with more definitive evidence should contact the FAQ maintainer.

5 4. Curve Computations

5.1 : How do I generate a Bezier curve that is parallel to another Bezier?

You can't. The only case where this is possible is when the Bezier can be represented by a straight line. And then the parallel 'Bezier' can also be represented by a straight line.

The situation is different for the broader class of rational Bezier curves. For example, these can represent circular arcs, and a parallel offset is just a concentric circular arc, also representable as a rational Bezier.

5.2 : How do I split a Bezier at a specific value for t?

A Bezier curve is a parametric polynomial function from the interval [0..1] to a space, usually 2-D or 3-D. Common Bezier curves use cubic polynomials, so have the form

f(t) = a3 t^3 + a2 t^2 + a1 t + a0,

where the coefficients are points in 3-D. Blossoming converts this polynomial to a more helpful form. Let s = 1-t, and think of tri-linear interpolation:

```
F([s0,t0],[s1,t1],[s2,t2]) =
    s0(s1(s2 c30 + t2 c21) + t1(s2 c21 + t2 c12)) +
    t0(s1(s2 c21 + t2 c12) + t1(s2 c12 + t2 c03))
    =
    c30(s0 s1 s2) +
    c21(s0 s1 t2 + s0 t1 s2 + t0 s1 s2) +
    c12(s0 t1 t2 + t0 s1 t2 + t0 t1 s2) +
    c03(t0 t1 t2).
```

The data points c30, c21, c12, and c03 have been used in such a way as to make the resulting function give the same value if any two arguments, say [s0,t0] and [s2,t2], are swapped. When [1-t,t] is used for all three arguments, the result is the cubic Bezier curve with those data points controlling it:

```
f(t) = F([1-t,t],[1-t,t],[1-t,t])
     =  (1-t)^3 c30 +
       3(1-t)^2 t c21 +
       3(1-t) t^2 c12 +
       t^3 c03.
```

Notice that

```
F([1,0],[1,0],[1,0]) = c30,
F([1,0],[1,0],[0,1]) = c21,
F([1,0],[0,1],[0,1]) = c12, _
F([0,1],[0,1],[0,1]) = c03.
```

In other words, cij is obtained by giving F argument t's i of which are 0 and j of which are 1. Since F is a linear polynomial in each argument, we can find f(t) using a series of simple steps. Begin with

f000 = c30, f001 = c21, f011 = c12, f111 = c03.

Then compute in succession:

```
f00t = s f000 + t f001, f01t = s f001 + t f011,
f11t = s f011 + t f111,
f0tt = s f00t + t f01t, f1tt = s f01t + t f11t,
fttt = s f0tt + t f1tt.
```

This is the de Casteljau algorithm for computing f(t) = fttt.

It also has split the curve for the intervals [0..t] and [t..1]. The control points for the first interval are f000, f00t, f0tt, fttt, while those for the second interval are fttt, f1tt, f11t, f111.

If you evaluate 3 F([1-t,t],[1-t,t],[-1,1]) you will get the derivate of f at t. Similarly, 2*3 F([1-t,t],[-1,1],[-1,1]) gives the second derivative of f at t, and finally using 1*2*3 F([-1,1],[-1,1],[-1,1]) gives the third derivative.

This procedure is easily generalized to different degrees, triangular patches, and B-spline curves.

5.3 : How do I find a t value at a specific point on a Bezier?

In general, you'll need to find t closest to your search point. There are a number of ways you can do this. Look at [Gems I, 607], there's a chapter on finding the nearest point on the Bezier curve. In my experience, digitizing the Bezier curve is an acceptable method. You can also try recursively subdividing the curve, see if you point is in the convex hull of the control points, and checking is the control points are close enough to a linear line segment and find the nearest point on the line segment, using linear interpolation and keeping track of the subdivision level, you'll be able to find t.

5.4 : How do I fit a Bezier curve to a circle?

Interestingly enough, Bezier curves can approximate a circle but not perfectly fit a circle.

Michael Goldapp, "Approximation of circular arcs by cubic polynomials" Computer Aided Geometric Design (#8 1991 pp.227-238)

Tor Dokken and Morten Daehlen, "Good Approximations of circles by curvature-continuous Bezier curves" Computer Aided Geometric Design (#7 1990 pp. 33-41).

6 5. 3D computations

6.1 : How do I rotate a 3D point?

Assuming you want to rotate vectors around the origin of your coordinate system. (If you want to rotate around some other point, subtract its coordinates from the point you are rotating, do the rotation, and then add back what you subtracted.) In 3-D, you need not only an angle, but also an axis. (In higher dimensions it gets much worse, very quickly.) Actually, you need 3 independent numbers, and these come in a variety of flavors. The flavor I recommend is unit quaternions: 4 numbers that square and add up to +1. You can write these as [(x,y,z),w], with 4 real numbers, or [v,w], with v, a 3-D vector pointing along the axis. The concept of an axis is unique to 3-D. It is a line through the origin containing all the points which do not move during the rotation. So we know if we are turning forwards or back, we use a vector pointing out along the line. Suppose you want to use unit vector u as your axis, and rotate by 2t degrees. (Yes, that's twice t.) Make a quaternion [u sin t, cos t]. You can use the quaternion – call it q – directly on a vector v with quaternion multiplication, q v q^-1, or just convert the quaternion to a 3x3 matrix M. If the components of q are {(x,y,z),w], then you want the matrix

M = {{1-2(yy+zz), 2(xy-wz), 2(xz+wy)}, { 2(xy+wz),1-2(xx+zz), 2(yz-wx)}, { 2(xz-wy), 2(yz+wx),1-2(xx+yy)}}.

Rotations, translations, and much more are explained in all basic computer graphics texts. Quaternions are covered briefly in [Foley], and more extensively in several Graphics Gems, and the SIGGRAPH 85 proceedings.

```
/* The following routine converts an angle and a unit axis vector
    * to a matrix, returning the corresponding unit quaternion at no
    * extra cost. It is written in such a way as to allow both fixed
    * point and floating point versions to be created by appropriate
    * definitions of FPOINT, ANGLE, VECTOR, QUAT, MATRIX, MUL, HALF,
    * TWICE, COS, SIN, ONE, and ZERO.
    * The following is an example of floating point definitions.
    #define FPOINT double
    #define ANGLE FPOINT
    #define VECTOR QUAT
    typedef struct {FPOINT x,y,z,w;} QUAT;
    enum Indices {X,Y,Z,W};
    typedef FPOINT MATRIX[4][4];
    #define MUL(a,b) ((a)*(b))
```

```
    #define HALF(a) ((a)*0.5)
    #define TWICE(a) ((a)*2.0)
    #define COS cos
    #define SIN sin
    #define ONE 1.0
    #define ZERO 0.0
*/
QUAT MatrixFromAxisAngle(VECTOR axis, ANGLE theta, MATRIX m)
{
    QUAT q;
    ANGLE halfTheta = HALF(theta);
    FPOINT cosHalfTheta = COS(halfTheta);
    FPOINT sinHalfTheta = SIN(halfTheta);
    FPOINT xs, ys, zs, wx, wy, wz, xx, xy, xz, yy, yz, zz;
    q.x = MUL(axis.x,sinHalfTheta);
    q.y = MUL(axis.y,sinHalfTheta);
    q.z = MUL(axis.z,sinHalfTheta);
    q.w = cosHalfTheta;
    xs = TWICE(q.x);  ys = TWICE(q.y);  zs = TWICE(q.z);
    wx = MUL(q.w,xs); wy = MUL(q.w,ys); wz = MUL(q.w,zs);
    xx = MUL(q.x,xs); xy = MUL(q.x,ys); xz = MUL(q.x,zs);
    yy = MUL(q.y,ys); yz = MUL(q.y,zs); zz = MUL(q.z,zs);
    m[X][X] = ONE - (yy + zz); m[X][Y] = xy - wz; m[X][Z] = xz + wy;
    m[Y][X] = xy + wz; m[Y][Y] = ONE - (xx + zz); m[Y][Z] = yz - wx;
    m[Z][X] = xz - wy; m[Z][Y] = yz + wx; m[Z][Z] = ONE - (xx + yy);
    /* Fill in remainder of 4x4 homogeneous transform matrix. */
    m[W][X] = m[W][Y] = m[W][Z] = m[X][W] = m[Y][W] = m[Z][W] = ZERO;
    m[W][W] = ONE;
    return (q);
}
/* The routine just given, MatrixFromAxisAngle, performs rotation about
    * an axis passing through the origin, so only a unit vector was needed
    * in addition to the angle. To rotate about an axis not containing the
    * origin, a point on the axis is also needed, as in the following. For
    * mathematical purity, the type POINT is used, but may be defined as:
    #define POINT VECTOR
*/
QUAT MatrixFromAnyAxisAngle(POINT o, VECTOR axis, ANGLE theta, MATRIX m)
{
    QUAT q;
    q = MatrixFromAxisAngle(axis,theta,m);
    m[X][W] = o.x-(MUL(m[X][X],o.x)+MUL(m[X][Y],o.y)+MUL(m[X][Z],o.z));
    m[Y][W] = o.y-(MUL(m[Y][X],o.x)+MUL(m[Y][Y],o.y)+MUL(m[Y][Z],o.z));
    m[Z][W] = o.x-(MUL(m[Z][X],o.x)+MUL(m[Z][Y],o.y)+MUL(m[Z][Z],o.z));
    return (q);
}
/* An axis can also be specified by a pair of points, with the direction
```

```
    * along the line assumed from the ordering of the points. Although both
    * the previous routines assumed the axis vector was unit length without
    * checking, this routine must cope with a more delicate possibility. If
    * the two points are identical, or even nearly so, the axis is unknown.
    * For now the auxiliary routine which makes a unit vector ignores that.
    * It needs definitions like the following for floating point.
    #define SQRT sqrt
    #define RECIPROCAL(a) (1.0/(a))
*/
VECTOR Normalize(VECTOR v)
{
    VECTOR u;
    FPOINT norm = MUL(v.x,v.x)+MUL(v.y,v.y)+MUL(v.z,v.z);
    /* Better to test for (near-)zero norm before taking reciprocal. */
    FPOINT scl = RECIPROCAL(SQRT(norm));
    u.x = MUL(v.x,scl); u.y = MUL(v.y,scl); u.z = MUL(v.z,scl);
    return (u);
}
QUAT MatrixFromPointsAngle(POINT o, POINT p, ANGLE theta, MATRIX m)
{
    QUAT q;
    VECTOR diff, axis;
    diff.x = p.x-o.x; diff.y = p.y-o.y; diff.z = p.z-o.z;
    axis = Normalize(diff);
    return (MatrixFromAnyAxisAngle(o,axis,theta,m));
}
```

6.2 : What is ARCBALL and where is the source?

Arcball is a general purpose 3-D rotation controller described by Ken Shoemake in the Graphics Interface '92 Proceedings. It features good behavior, easy implementation, cheap execution, and optional axis constraints. A Macintosh demo and electronic version of the original paper (Microsoft Word format) may be obtained from ftp::/ftp.cis.upenn.edu/pub/graphics.

Complete source code appears in Graphics Gems IV pp. 175-192. A fairly complete sketch of the code appeared in the original article, in Graphics Interface 92 Proceedings, available from Morgan Kaufmann.

6.3 : How do I clip a polygon against a rectangle?

This is the Sutherland-Hodgman algorithm, an old favorite that should be covered in any textbook. Any of the references listed in the FAQ should have it. According to Vatti (q.v.) "This [algorithm] produces degenerate edges in certain concave / self intersecting polygons that need to be removed as a special extension to the main algorithm" (though this is not a problem if all you are doing with the results is scan converting them.)

6.4 : How do I clip a polygon against another polygon?

Klamer Schutte, klamer@ph.tn.tudelft.nl has developed and implemented some code in C++ to perform clipping of two possibly concave 2-D polygons. A description can be found at: *http://www.ph.tn.tudelft.nl*:/People/klamer/clippoly_entry.html To compile the source code you will need a C++ compiler with templates, such as g++. The source code is available at: *ftp://ftp.ph.tn.tudelft.nl/pub/klamer/clippoly.tar.gz* See also *http://members.xoom.com/msleonov/pbcomp.html*, which extends the above to permit holes.

Alan Murta released a polygon clipper library (in C) which uses a modified version of the Vatti algorithm: *http://www.cs.man.ac.uk/aig/staff/alan/software/index.html*

References:

Weiler, K. "Polygon Comparison Using a Graph Representation", SIGGRAPH 80 pg. 10-18

Vatti, Bala R. "A Generic Solution to Polygon Clipping", Communications of the ACM, July 1992, Vol 35, No. 7, pg. 57-63

6.5 : How do I find the intersection of a line and a plane?

If the plane is defined as:

a*x + b*y + c*z + d = 0

and the line is defined as:

```
x = x1 + (x2 - x1)*t = x1 + i*t
y = y1 + (y2 - y1)*t = y1 + j*t
z = z1 + (z2 - z1)*t = z1 + k*t
```

Then just substitute these into the plane equation. You end up with:

t = - (a*x1 + b*y1 + c*z1 + d)/(a*i + b*j + c*k)

When the denominator is zero, the line is contained in the plane if the numerator is also zero (the point at t=0 satisfies the plane equation), otherwise the line is parallel to the plane.

6.6 : How do I determine the intersection between a ray and a triangle?

First find the intersection between the ray and the plane in which the triangle is situated. Then see if the point of intersection is inside the triangle. Details may be found in [O'Rourke (C)] pp.226-238, whose code is at *http://cs.smith.edu/* orourke/ . Efficient code complete with statistical tests is described in the Mo:ller- Trumbore paper in J. Graphics Tools (C code downloadable from there): *http://www.acm.org/jgt/papers/MollerTrumbore97/* See also the full paper: *http://www.Graphics.Cornell.EDU/pubs/1997/MT97.html*

6.7 : How do I determine the intersection between a ray and a sphere

Given a ray defined as:

```
x = x1 + (x2 - x1)*t = x1 + i*t
y = y1 + (y2 - y1)*t = y1 + j*t
z = z1 + (z2 - z1)*t = z1 + k*t
```

and a sphere defined as:

(x - l)**2 + (y - m)**2 + (z - n)**2 = r**2

Substituting in gives the quadratic equation:

a*t**2 + b*t + c = 0

where:

```
a = i**2 + j**2 + k**2
b = 2*i*(x1 - l) + 2*j*(y1 - m) + 2*k*(z1 - n)
c = (x1-l)**2 + (y1-m)**2 + (z1-n)**2 - r**2;
```

If the discriminant of this equation is less than 0, the line does not intersect the sphere. If it is zero, the line is tangential to the sphere and if it is greater than zero it intersects at two points. Solving the equation and substituting the values of t into the ray equation will give you the points.

Reference:

[Glassner:RayTracing]

6.8 : How do I find the intersection of a ray and a Bezier surface?

Joy I.K. and Bhetanabhotla M.N., "Ray tracing parametric surfaces utilizing numeric techniques and ray coherence", Computer Graphics, 16, 1986, 279-286

Joy and Bhetanabhotla is only one approach of three major method classes: tessellation, direct computation, and a hybrid of the two. Tessellation is relatively easy (you intersect the polygons making up the tessellation) and has no numerical problems, but can be inaccurate; direct is cheap on memory, but very expensive computationally, and can (and usually does) suffer from precision problems within the root solver; hybrids try to blend the two.

Reference:

[Glassner:RayTracing]

6.9 : How do I ray trace caustics?

See the work of Henrik Wann Jensen at *http://www.gk.dtu.dk/* hwj/papers/gi98.html | [Link stale – can anyone update? -jor/1Dec99]

```
@inproceedings{j-rcnls-96
, author =      "Henrik Wann Jensen"
, title =       "Rendering Caustics on Non-Lambertian Surfaces"
, booktitle =   "Proc. Graphics Interface '96"
```

```
, pages =       "116--121"
, location =    "Toronto"
, year =         1996
}
```

Some older references:

An expensive answer: @InProceedings{mitchell-1992-illumination, author = "Don P. Mitchell and Pat Hanrahan", title = "Illumination From Curved Reflectors", year = "1992", month = "July", volume = "26", booktitle = "Computer Graphics (SIGGRAPH '92 Proceedings)", pages = "283–291", keywords = "caustics, interval arithmetic, ray tracing", editor = "Edwin E. Catmull", }

A cheat: @Article{inakage-1986-caustics, author = "Masa Inakage", title = "Caustics and Specular Reflection Models for Spherical Objects and Lenses ", pages = "379–383", journal = "The Visual Computer", volume = "2", number = "6", year = "1986", keywords = "ray tracing effects", }

Very specialized: @Article{yuan-1988-gemstone, author = "Ying Yuan and Tosiyasu L. Kunii and Naota Inamato and Lining Sun ", title = "Gemstone Fire: Adaptive Dispersive Ray Tracing of Polyhedrons", year = "1988", month = "November", journal = "The Visual Computer", volume = "4", number = "5", pages = "259–70", keywords = "caustics", }

6.10 : What is the marching cubes algorithm?

The marching cubes algorithm is used in volume rendering to construct an isosurface from a 3D field of values.

The 2D analog would be to take an image, and for each pixel, set it to black if the value is below some threshold, and set it to white if it's above the threshold. Then smooth the jagged black outlines by skinning them with lines.

The marching cubes algorithm tests the corner of each cube (or voxel) in the scalar field as being either above or below a given threshold. This yields a collection of boxes with classified corners. Since there are eight corners with one of two states, there are 256 different possible combinations for each cube. Then, for each cube, you replace the cube with a surface that meets the classification of the cube. For example, the following are some 2D examples showing the cubes and their associated surface.

```
- ----- +      - ----- -      - ----- +      - ----- +
|:::'   |      |:::::::|      |::::   |      |   ':::|
|:'     |      |:::::::|      |::::   |      |.    ':|
|       |      |       |      |::::   |      |::.    |
+ ----- +      + ----- +      - ----- +      + ----- -
```

The result of the marching cubes algorithm is a smooth surface that approximates the isosurface that is constant along a given threshold. This is useful for displaying a volume of oil in a geological volume, for example.

References: "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", William E. Lorensen and Harvey E. Cline, Computer Graphics (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, pp. 163-169.

[Watt:Animation] pp. 302-305 and 313-321 [Schroeder] James Sharman's description: *http://www.exaflop.org/docs/marchcubes*

6.11 : What is the status of the patent on the "marching cubes" algorithm?

United States Patent Number: 4,710,876 Date of Patent: Dec. 1, 1987 Inventors: Harvey E. Cline, William E. Lorensen Assignee: General Electric Company Title: "System and Method for the Display of Surface Structures Contained Within the Interior Region of a Solid Body" Filed: Jun. 5, 1985 *http://www.patents.ibm.com/details*?patent_number=4710876

United States Patent Number: 4,885,688 Date of Patent: Dec. 5, 1989 Inventor: Carl R. Crawford Assignee: General Electric Company Title: "Minimization of Directed Points Generated in Three-Dimensional Dividing Cubes Method" Filed: Nov. 25, 1987 *http://www.patents.ibm.com/details*?patent_number=4885688

6.12 : How do I do a hidden surface test (backface culling) with 3d points?

Just define all points of all polygons in clockwise order.

```
c = (x3*((z1*y2)-(y1*z2))+
    (y3*((x1*z2)-(z1*x2))+
    (z3*((y1*x2)-(x1*y2))+
```

x1,y1,z1, x2,y2,z2, x3,y3,z3 = the first three points of the polygon.

If c is positive the polygon is visible. If c is negative the polygon is invisible (or the other way).

6.13 : Where can I find algorithms for 3D collision detection?

Check out "proxima", from Purdue, which is a C++ library for 3D collision detection for arbitrary polyhedra. It's a nice system; the algorithms are sophisticated, but the code is of modest size.

ftp://ftp.cs.purdue.edu/pub/pse/PROX/prox9.1.tar.Z

For information about the I_COLLIDE 3D collision detection system *http://www.cs.unc.edu/* geom/I_COLLIDE.html

"Fast Collision Detection of Moving Convex Polyhedra", Rich Rabbitz, Graphics Gems IV, pages 83-109, includes source in C.

SOLID: "a library for collision detection of three-dimensional objects undergoing rigid motion and deformation. SOLID is designed to be used in interactive 3D graphics applications, and is especially suited for collision detection of objects and worlds described in VRML. Written in standard C++, compiles under GNU g++ version 2.8.1 and Visual C++ 5.0." See: *http://www.win.tue.nl/cs/tt/gino/solid/*

6.14 : How do I perform basic viewing in 3d?

We describe the shape and position of objects using numbers, usually (x,y,z) coordinates. For example, the corners of a cube might be {(0,0,0), (1,0,0), (1,1,0), (0,1,0), (0,0,1), (1,0,1), (1,1,1), (0,1,1)}. A deep understanding of what we are saying with these numbers requires mathematical study, but I will try to keep this simple. At the least, we must understand that we have designated some point in space as the origin–coordinates (0,0,0)–and marked off lines in 3 mutually perpendicular directions using equally spaced units to give us (x,y,z) values. It might be helpful to know if we are using 1 to mean 1 foot, 1 meter, or 1 parsec; the numbers alone do not tell us.

A picture on a screen is two steps removed from the 3D world it depicts. First, it is a 2D projection; and second, it is a finite resolution approximation to the infinitely precise projection. I will ignore the approximation (sampling) for now. To know what the projection looks like, we need to know where our viewpoint is, and where the plane of the projection is, both in the 3D world. Think of it as looking out a window into a scene. As artists discovered some 500 years ago, each point in the world appears to be at a point on the window. If you move your head or look out a different window, everything changes. When the mathematicians understood what the artists were doing, they invented perspective geometry.

If your viewpoint is at the origin–(0,0,0)–and the window sits parallel to the x-y plane but at z=1, projection is no more than (x,y,z) in the world appears at (x/z,y/z,1) on the plane. Distant points will have large z values, causing them to shrink in the picture. That's perspective.

The trick is to take an arbitrary viewpoint and plane, and transform the world so we have the simple viewing situation. There are two steps: move the viewpoint to the origin, then move the viewplane to the z=1 plane. If the viewpoint is at (vx,vy,vz), transform every point by the translation (x,y,z) –> (x-vx,y-vy,z-vz). This includes the viewpoint and the viewplane. Now we need to rotate so that the z axis points straight at the viewplane, then scale so it is 1 unit away.

After all this, we may find ourselves looking out upside- down. It is traditional to specify some direction in the world or viewplane as "up", and rotate so the positive y axis points that way (as nearly as possible if it's a world vector). Finally, we have acted so far as if the window was the entire plane instead of a limited portal. A final shift and scale transforms coordinates in the plane to coordinates on the screen, so that a rectangular region of interest (our "window") in the plane fills a rectangular region of the screen (our "canvas" if you like).

I have left out details of how you define and perform the rotation of the viewplane, but I'm sure someone else will be happy to supply those if there is demand. It requires knowing how to describe a plane, and how to rotate vectors to line up. Neither is difficult, but this is already using a lot of net space. One further practical difficulty is the need to clip away parts of the world behind us, so -(x,y,z) doesn't pop up at (x/z,y/z,1). (Notice the mathematics of projection alone would allow that!) But all the viewing transformations can be done using translation, rotation, scale, and a final perspective divide. If a 4x4 homogeneous matrix is used, it can represent everything needed, which saves a lot of work.

6.15 : How do I optimize/simplify a 3D polygon mesh

References: "Mesh Optimization" Hoppe, DeRose Duchamp, McDonald, Stuetzle, ACM COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1993.

"Re-Tiling Polygonal Surfaces", Greg Turk, ACM Computer Graphics, 26, 2, July 1992

"Decimation of Triangle Meshes", Schroeder, Zarge, Lorensen, ACM Computer Graphics, 26, 2 July 1992

"Simplification of Objects Rendered by Polygonal Approximations", Michael J. DeHaemer, Jr. and Michael J. Zyda, Computer & Graphics, Vol. 15, No. 2, 1991, Great Britain: Pergamon Press, pp. 175-184.

"Topological Refinement Procedures for Triangular Finite Element Procedures", S. A. Cannan, S. N. Muthukrishnan and R. P. Phillips, Engineering With Computers, No. 12, p. 243-255, 1996.

"Progressive Meshes", Hoppe, SIGGRAPH 96, *http://research.microsoft.com/* hoppe/siggraph96pm/paper.htm

[This list is in need of updating. -jor]

6.16 : How can I perform volume rendering?

Two principal methods can be used: - Ray casting or front-to-back, where the volume is behind the projection plane. A ray is projected from each point in the projection plane through the volume. The ray accumulates the properties of each voxel it passes through. - Object order or back-to-front, where the projection plane is behind the volume. Each slice of the volume is projected on the projection plane, from the farest plane to the nearest plane.

You can also use the marching-cubes algorithm, if the extraction of surfaces from the data set is easy to do (see Subject 5.10).

Here is one algorithm to do front-to-back volume rendering:

Set up a projection plane as a plane tangent to a sphere that encloses the volume. From each pixel of the projection plane, cast a ray through the volume by using a Bresenham 3D algorithm. The ray accumulates properties from each voxel intersected, stopping when the ray exits the volume. The pixel value on the projection plane determines the final color of the ray.

For unshaded images (i.e., without gradient and light computations),

```
if  C  is the ray color           t  the ray transparency
    C' the new ray color          t' the new ray transparency
    Cv the voxel color            tv the voxel transparency
then:
    C' = C + t*Cv       and       t' = t * tv
with initial values: C = 0.0 and t = 1.0
```

An alternate version: instead of C' = C + t*Cv , use : C' = C + t*Cv*(1-tv)^p with p a float variable. Sometimes this gives the best results. When the ray has accumulated transparency, if it becomes negligible (e.g., t<0.1), the process can stop and proceed to the next ray.

References:

Bresenham 3D: - *http://www.sct.gu.edu.au/* anthony/info/graphics/bresenham.procs - [Gems IV] p. 366 Volume rendering: - [Watt:Animation] pp. 297-321 - IEEE Computer Graphics and application Vol. 10, Nb. 2, March 1990 - pp. 24-53 - "Volume Visualization" Arie Kaufman - Ed. IEEE Computer Society Press Tutorial - "Efficient Ray Tracing of Volume Data" Marc Levoy - ACM Transactions on Graphics, Vol. 9, Nb 3, July 1990

6.17 : Where can I get the spline description of the famous teapot etc.?

See the Standard Procedural Databases software, whose official distribution site is *http://www.acm.org/tog/resources/SPD/* This database contains much useful 3D code, including spline surface tessellation, for the teapot.

6.18 : How can the distance between two lines in space be computed?

The following is robust C code from Seth Teller that computes the "points of closest approach" on two 3D lines. It also classifies the lines as parallel, intersecting, or (the generic case) skew. What's listed below shows the main ideas; the full code is at *http://graphics.lcs.mit.edu/* seth/geomlib/linelinecp.c

```
// computes pB ON line B closest to line A
// computes pA ON line A closest to line B
// return: 0 if parallel; 1 if coincident; 2 if generic (i.e., skew)
int
line_line_closest_points3d (
    POINT *pA, POINT *pB,                   // computed points
    const POINT *a, const VECTOR *adir,             // line A, point-normal form
    const POINT *b, const VECTOR *bdir )    // line B, point-normal form
{
static VECTOR Cdir, *cdir =
static PLANE Ac, *ac = , Bc, *bc =

    // connecting line is perpendicular to both
    vcross ( cdir, adir, bdir );

    // check for near-parallel lines
    if ( !vnorm( cdir ) )   {
        *pA = *a;   // all points are closest
        *pB = *b;
        return 0;   // degenerate: lines parallel
        }

    // form plane containing line A, parallel to cdir
    plane_from_two_vectors_and_point ( ac, cdir, adir, a );

    // form plane containing line B, parallel to cdir
    plane_from_two_vectors_and_point ( bc, cdir, bdir, b );

    // closest point on A is line A ^ bc
    intersect_line_plane ( pA, a, adir, bc );

    // closest point on B is line B ^ ac
    intersect_line_plane ( pB, b, bdir, ac );

    // distinguish intersecting, skew lines
    if ( edist( pA, pB ) < 1.0E-5F )
        return 1;     // coincident: lines intersect
    else
        return 2;     // distinct: lines skew
}
```

Also Dave Eberly has code for computing distance between various geometric primitives, including MinLine-Line(), at *http://www.magic*-software.com

6.19 : How can I compute the volume of a polyhedron?

Assume that the surface is closed, every face is a triangle, and the vertices of each triangle are oriented ccw from the outside. Let Volume(t,p) be the signed volume of the tetrahedron formed by a point p and a triangle t. This may be computed by a 4x4 determinant, as in [O'Rourke (C), p.26]. Choose an arbitrary point p (e.g., the origin), and compute the sum Volume(t_i,p) for every triangle t_i of the surface. That is the volume of the object. The justification for this claim is nontrivial, but is essentially the same as the justification for the computation of the area of a polygon (Subject 2.01).

C Code available at *http://cs.smith.edu/* orourke/ and *http://www.acm.org/jgt/papers/Mirtich96/* .

For computing the volumes of n-d convex polytopes, there is a C implementation by Bueeler and Enge of various algorithms available at

http://www.Mathpool.Uni-Augsburg.DE/ enge/Volumen.html .

6.20 : How can I decompose a polyhedron into convex pieces?

Usually this problem is interpreted as seeking a collection of pairwise disjoint convex polyhedra whose set union is the original polyhedron P. The following facts are known about this difficult problem:

o Not every polyhedron may be partitioned by diagonals into tetrahedra. A counterexample is due to Scho:nhardt [O'Rourke (A), p.254].

o Determining whether a polyhedron may be so partitioned is NP-hard, a result due to Seidel & Ruppert [Disc. Comput. Geom. 7(3) 227-254 (1992).]

o Removing the restriction to diagonals, i.e., permitting so-called Steiner points, there are polyhedra of n vertices that require n^2 convex pieces in any decomposition. This was established by Chazelle [SIAM J. Comput. 13: 488-507 (1984)]. See also [O'Rourke (A), p.256]

o An algorithm of Palios & Chazelle guarantees at most O(n+r^2) pieces, where r is the number of reflex edges (i.e., grooves). [Disc. Comput. Geom. 5:505-526 (1990).]

o Barry Joe's geompack package, at *ftp://ftp.cs.ualberta.ca/pub/geompack*, includes a 3D convex decomposition Fortran program.

o There seems to be no other publicly available code.

6.21 : How can the circumsphere of a tetrahedron be computed?

Let a, b, c, and d be the corners of the tetrahedron, with ax, ay, and az the components of a, and likewise for b, c, and d. Let |a| denote the Euclidean norm of a, and let a x b denote the cross product of a and b. Then the center m and radius r of the circumsphere are given by

```
|                                                                     |
| |d-a|^2 [(b-a)x(c-a)] + |c-a|^2 [(d-a)x(b-a)] + |b-a|^2 [(c-a)x(d-a)] |
|                                                                     |
```

```
r= ------------------------------------------------------------------------
                          | bx-ax  by-ay  bz-az |
                        2 | cx-ax  cy-ay  cz-az |
                          | dx-ax  dy-ay  dz-az |


and

       |d-a|^2 [(b-a)x(c-a)] + |c-a|^2 [(d-a)x(b-a)] + |b-a|^2 [(c-a)x(d-a)]
m= a + --------------------------------------------------------------------
                            | bx-ax  by-ay  bz-az |
                          2 | cx-ax  cy-ay  cz-az |
                            | dx-ax  dy-ay  dz-az |
```

Some notes on stability:

- Note that the expression for r is purely a function of differences between coordinates. The advantage is that the relative error incurred in the computation of r is also a function of the _differences_ between the vertices, and is not influenced by the _absolute_ coordinates of the vertices. In most applications, vertices are usually nearer to each other than to the origin, so this property is advantageous.

Similarly, the formula for m incurs roundoff error proportional to the differences between vertices, but not proportional to the absolute coordinates of the vertices until the final addition.

- These expressions are unstable in only one case: if the denominator is close to zero. This instability, which arises if the tetrahedron is nearly degenerate, is unavoidable. Depending on your application, you may want to use exact arithmetic to compute the value of the determinant. See *http://www.geom.umn.edu/software/cglist/alg.html* or *http://www.cs.cmu.edu/* quake/robust.html

6.22 : How do I determine if two triangles in 3D intersect?

Let the two triangles be T1 and T2. If T1 lies strictly to one side of the plane containing T2, or T2 lies strictly to one side of the plane containing T1, the triangles do not intersect. Otherwise, compute the line of intersection L between the planes. Let Ik be the interval (Tk inter L), k=1,2. Either interval may be empty. T1 and T2 intersect iff I1 and I2 overlap.

This method is decribed in Tomas Mo:ller, "A fast triangle-triangle intersection test," J. Graphics Tools 2(2) 25-30 1997. C code at *http://www.acm.org/jgt/papers/Moller97/*

6.23 : How can a 3D surface be reconstructed from a collection of points?

This is a difficult problem. There are two main variants: (1) when the points are organized into parallel slices through the object; (2) when the points are unorganized. For (1), see this survey: D. Meyers, S. Skinner, K. Sloan. "Surfaces from Contours" ACM Trans. Graph. 11(3) Jul 1992, 228–258. Code is available at *http://www*-sop.inria.fr/prisme/personnel/geiger/nuages.html For (2), see this paper: H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle "Surface reconstruction from unorganized points" Proc. SIGGRAPH '92, 71–78. and P. Kumar's collection of links at *http://members.tripod.com/* GeomWiz/www.sites.html

7 6. Geometric Structures and Mathematics

7.1 : Where can I get source for Voronoi/Delaunay triangulation?

For 2-d Delaunay triangulation, try Shewchuk's triangle program. It includes options for constrained triangulation and quality mesh generation. It uses exact arithmetic.

The Delaunay triangulation is equivalent to computing the convex hull of the points lifted to a paraboloid. For n-d Delaunay triangulation try Clarkson's hull program (exact arithmetic) or Barber and Huhdanpaa's Qhull program (floating point arithmetic). The hull program also computes Voronoi volumes and alpha shapes. The Qhull program also computes 2-d Voronoi diagrams and n-d Voronoi vertices. The output of both programs may be visualized with Geomview.

There are many other codes for Delaunay triangulation and Voronoi diagrams. See Amenta's list of computational geometry software.

The Delaunay triangulation satisfies the following property: the circumcircle of each triangle is empty. The Voronoi diagram is the closest-point map, i.e., each Voronoi cell identifies the points that are closest to an input site. The Voronoi diagram is the dual of the Delaunay triangulation. Both structures are defined for general dimension. Delaunay triangulation is an important part of mesh generation.

There is a FAQ of polyhedral computation explaining how to compute n-d Delaunay triangulation and n-d Voronoi diagram using a convex hull code, and how to use the linear programming technique to determine the Voronoi cells adjacent to a given Voronoi cell efficiently for large scale or higher dimensional cases.

Avis' lrs code uses the same file formats as cdd. It uses exact arithmetic and is useful for problems with very large output size, since it does not require storing the output.

References:

Amenta: *http://www.geom.umn.edu/software/cglist*

Avis: *ftp://mutt.cs.mcgill.ca/pub/C/lrs.html*

Barber & *http://www.geom.umn.edu/locate/qhull* Huhdanpaa *ftp://geom.umn.edu/pub/software/qhull.tar.Z*

Clarkson: *http://cm.bell*-labs.com/netlib/voronoi/hull.html *ftp://cm.bell*-labs.com/netlib/voronoi/hull.shar.Z

Geomview: *http://www.geom.umn.edu/locate/geomview* *ftp://geom.umn.edu/pub/software/geomview/*

Polyhedral Computation FAQ: *http://www.ifor.math.ethz.ch/ifor/staff/fukuda/fukuda.html*

Shewchuk: *http://www.cs.cmu.edu/* quake/triangle.html *ftp://cm.bell*-labs.com/netlib/voronoi/triangle.shar.Z

[O' Rourke (C)] pp. 168-204

[Preparata & Shamos] pp. 204-225

Chew, L. P. 1987. "Constrained Delaunay Triangulations," Proc. Third Annual ACM Symposium on Computational Geometry.

Chew, L. P. 1989. "Constrained Delaunay Triangulations," Algorithmica 4:97-108. (UPDATED VERSION OF CHEW 1987.)

Fang, T-P. and L. A. Piegl. 1994. "Algorithm for Constrained Delaunay Triangulation," The Visual Computer 10:255-265. (RECOMMENDED!)

Frey, W. H. 1987. "Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes," International Journal for Numerical Methods in Engineering 24:2183-2200.

Guibas, L. and J. Stolfi. 1985. "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," ACM Transactions on Graphics 4(2):74-123.

Karasick, M., D. Lieber, and L. R. Nackman. 1991. "Efficient Delaunay Triangulation Using Rational Arithmetic," ACM Transactions on Graphics 10(1):71-91.

Lischinski, D. 1994. "Incremental Delaunay Triangulation," Graphics Gems IV, P. S. Heckbert, Ed. Cambridge, MA: Academic Press Professional. (INCLUDES C++ SOURCE CODE – THANK YOU, DANI!)

[Okabe]

Schuierer, S. 1989. "Delaunay Triangulation and the Radiosity Approach," Proc. Eurographics '89, W. Hansmann, F. R. A. Hopgood, and W. Strasser, Eds. Elsevier Science Publishers, 345-353.

Subramanian, G., V. V. S. Raveendra, and M. G. Kamath. 1994. "Robust Boundary Triangulation and Delaunay Triangulation of Arbitrary Triangular Domains," International Journal for Numerical Methods in Engineering 37(10):1779-1789.

Watson, D. F. and G. M. Philip. 1984. "Survey: Systematic Triangulations," Computer Vision, Graphics, and Image Processing 26:217-223.

7.2 : Where do I get source for convex hull?

For n-d convex hulls, try Clarkson's hull program (exact arithmetic) or Barber and Huhdanpaa's Qhull program (floating point arithmetic). Qhull handles numeric precision problems by merging facets. The output of both programs may be visualized with Geomview.

In higher dimensions, the number of facets may be much smaller than the number of lower-dimensional features. When this is the case, Fukuda's cdd program is much faster than Qhull or hull.

There are many other codes for convex hulls. See Amenta's list of computational geometry software.

References:

Amenta: *http://www.geom.umn.edu/software/cglist/*

Barber & *http://www.geom.umn.edu/locate/qhull* Huhdanpaa *ftp://geom.umn.edu/pub/software/qhull.tar.Z*

Clarkson: *http://cm.bell*-labs.com/netlib/voronoi/hull.html *ftp://cm.bell*-labs.com/netlib/voronoi/hull.shar.Z

Geomview: *http://www.geom.umn.edu/locate/geomview ftp://geom.umn.edu/pub/software/geomview/*

Fukuda: *http://www.ifor.math.ethz.ch/staff/fukuda/cdd_home/cdd.html ftp://ftp.ifor.math.ethz.ch/pub/fukuda/cdd/*

[O' Rourke (C)] pp. 70-167 C code for Graham's algorithm on p.80-96. Three-dimensional convex hull discussed in Chapter 4 (p.113-67). C code for the incremental algorithm on p.130-60.

[Preparata & Shamos] pp. 95-184

7.3 : Where do I get source for halfspace intersection?

For n-d halfspace intersection, try Fukuda's cdd program or Barber and Huhdanpaa's Qhull program. Both use floating point arithmetic. Fukuda includes code for exact arithmetic. Qhull handles numeric precision problems by merging facets.

Qhull computes halfspace intersection by computing a convex hull. The intersection of halfspaces about the origin is equivalent to the convex hull of the halfspace coefficients divided by their offsets.

References:

Barber & *http://www.geom.umn.edu/locate/qhull* Huhdanpaa *ftp://geom.umn.edu/pub/software/qhull.tar.Z*

Fukuda: *ftp://ifor13.ethz.ch/pub/fukuda/cdd/*

[Preparata & Shamos] pp. 315-320

7.4 : What are barycentric coordinates?

Let p1, p2, p3 be the three vertices (corners) of a closed triangle T (in 2D or 3D or any dimension). Let t1, t2, t3 be real numbers that sum to 1, with each between 0 and 1: t1 + t2 + t3 = 1, 0 <= ti <= 1. Then the point p = t1*p1 + t2*p2 + t3*p3 lies in the plane of T and is inside T. The numbers (t1,t2,t3) are called the barycentric coordinates of p with respect to T. They uniquely identify p. They can be viewed as masses placed at the vertices whose center of gravity is p. For example, let p1=(0,0), p2=(1,0), p3=(3,2). The barycentric coordinates (1/2,0,1/2) specify the point p = (0,0)/2 + 0*(1,0) + (3,2)/2 = (3/2,1), the midpoint of the p1-p3 edge. If p is joined to the three vertices, T is partitioned into three triangles. Call them T1, T2, T3, with Ti not incident to pi. The areas of these triangles Ti are proportional to the barycentric coordinates ti of p.

Reference: [Coxeter, Intro. to Geometry, p.217].

7.5 : How can I generate a random point inside a triangle?

Use barycentric coordinates (see item 53) . Let A, B, C be the three vertices of your triangle. Any point P inside can be expressed uniquely as P = aA + bB + cC, where a+b+c=1 and a,b,c are each >= 0. Knowing a and b permits you to calculate c=1-a-b. So if you can generate two random numbers a and b, each in [0,1], such that their sum <=1, you've got a random point in your triangle. Generate random a and b independently and uniformly in [0,1] (just divide the standard C rand() by its max value to get such a random number.) If a+b>1, replace a by 1-a, b by 1-b. Let c=1-a-b. Then aA + bB + cC is uniformly distributed in triangle ABC: the reflection step a=1-a; b=1-b gives a point (a,b) uniformly distributed in the triangle (0,0)(1,0)(0,1), which is then mapped affinely to ABC. Now you have barycentric coordinates a,b,c. Compute your point P = aA + bB + cC.

Reference: [Gems I], Turk, pp. 24-28.

7.6 : How do I evenly distribute N points on (tesselate) a sphere?

"Evenly distributed" doesn't have a single definition. There is a sense in which only the five Platonic solids achieve regular tesselations, as they are the only ones whose faces are regular and equal, with each vertex incident to the same number of faces. But generally "even distribution" focusses not so much on the induced tesselation, as it does on the distances and arrangement of the points/vertices. For example, eight points can be placed on the sphere further away from one another than is achieved by the vertices of an inscribed cube: start with an inscribed cube, twist the top face 45 degrees about the north pole, and then move the top and bottom points along the surface towards the equator a bit.

The various definitions of "evenly distributed" lead into moderately complex mathematics. This topic happens to be a FAQ on sci.math as well as on comp.graphics.algorithms; a much more extensive and rigorous discussion written by Dave Rusin can be found at: *http://www.math.niu.edu/* rusin/known-math/95/sphere.faq

A simple method of tesselating the sphere is repeated subdivision. An example starts with a unit octahedron. At each stage, every triangle in the tesselation is divided into 4 smaller triangles by creating 3 new vertices halfway along the edges. The new vertices are normalized, "pushing" them out to lie on the sphere. After N steps of subdivision, there will be 8 * 4^N triangles in the tesselation.

A simple example of tesselation by subdivision is available at *ftp://ftp.cs.unc.edu/pub/users/leech/sphere.c*

One frequently used definition of "evenness" is a distribution that minimizes a 1/r potential energy function of all the points, so that in a global sense points are as "far away" from each other as possible. Starting from an arbitrary distribution, we can either use numerical minimization algorithms or point repulsion, in which all the points are considered to repel each other according to a 1/r^2 force law and dynamics are simulated. The algorithm is run until some convergence criterion is satisfied, and the resulting distribution is our approximation.

Jon Leech developed code to do just this. It is available at *ftp://ftp.cs.unc.edu/pub/users/leech/points.tar.gz.* See his README file for more information. His distribution includes sample output files for various n < 300, which may be used off-the-shelf if that is all you need.

Another method that is simpler than the above, but attains less uniformity, is based on spacing the points along a spiral that encircles the sphere. Code available from links at *http://cs.smith.edu/* orourke/.

7.7 : What are coordinates for the vertices of an icosohedron?

Data on various polyhedra is available at *http://cm.bell*-labs.com/netlib/polyhedra/index.html, or *http://netlib.bell*-labs.com/netlib/polyhedra/index.html, or *http://www.netlib.org/polyhedra/index.html* For the icosahedron, the twelve vertices are:

(+- 1, 0, +-t), (0, +-t, +-1), and (+-t, +-1, 0)

```
where t = (sqrt(5)-1)/2, the golden ratio.
Reference: "Beyond the Third Dimension" by Banchoff, p.168.
```

7.8 : How do I generate random points on the surface of a sphere?

There are several methods. Perhaps the easiest to understand is the "rejection method": generate random points in an origin- centered cube with opposite corners (r,r,r) and (-r,-r,-r). Reject any point p that falls outside of the sphere of radius r. Scale the vector to lie on the surface. Because the cube to sphere volume ratio is pi/6, the average number of iterations before an acceptable vector is found is 6/pi = 1.90986. This essentially doubles the effort, and makes this method slower than the "trig method." A timing comparison conducted by Ken Sloan showed that the trig method runs in about 2/3's the time of the rejection method. He found that methods based on the use of normal distributions are twice as slow as the rejection method.

The trig method. This method works only in 3-space, but it is very fast. It depends on the slightly counterintuitive fact (see proof below) that each of the three coordinates is uniformly distributed on [-1,1] (but the three are not independent, obviously). Therefore, it suffices to choose one axis (Z, say) and generate a uniformly distributed value on that axis. This constrains the chosen point to lie on a circle parallel to the X-Y plane, and the obvious trig method may be used to obtain the remaining coordinates.

```
(a) Choose z uniformly distributed in [-1,1].
(b) Choose t uniformly distributed on [0, 2*pi).
(c) Let r = sqrt(1-z^2).
(d) Let x = r * cos(t).
(e) Let y = r * sin(t).
```

This method uses uniform deviates (faster to generate than normal deviates), and no set of coordinates is ever rejected.

Here is a proof of correctness for the fact that the z-coordinate is uniformly distributed. The proof also works for the x- and y- coordinates, but note that this works only in 3-space.

The area of a surface of revolution in 3-space is given by

A = 2 * pi * int_a^b f(x) * sqrt(1 + [f'(x}]^2) dx

Consider a zone of a sphere of radius R. Since we are integrating in the z direction, we have

```
f(z) = sqrt(R^2 - z^2)
f'(z) = -z / sqrt(R^2-z^2)

1 + [f'(z)]^2 = r^2 / (R^2-z^2)

A = 2 * pi * int_a^b sqrt(R^2-z^2) * R/sqrt(R^2-z^2) dz
  = 2 * pi * R int_a^b dz
  = 2 * pi * R * (b-a)
  = 2 * pi * R * h,
```

where h = b-a is the vertical height of the zone. Notice how the integrand reduces to a constant. The density is therefore uniform.

Here is simple C code implementing the trig method:

```
void SpherePoints(int n, double X[], double Y[], double Z[])
{
  int i;
  double x, y, z, w, t;

  for( i=0; i< n; i++ ) {
    z = 2.0 * drand48() - 1.0;
    t = 2.0 * M_PI * drand48();
    w = sqrt( 1 - z*z );
    x = w * cos( t );
```

```
        y = w * sin( t );
        printf("i=%d:  x,y,z=\t%10.5lf\t%10.5lf\t%10.5lf\n", i, x,y,z);
        X[i] = x; Y[i] = y; Z[i] = z;
      }
    }
```

A complete package is available at *ftp://cs.smith.edu/pub/code/sphere.tar.gz* (4K), reachable from *http://cs.smith.edu/* orourke/ .

7.9 : What are Plucker coordinates?

A common convention is to write umlauted u as "ue", so you'll also see "Pluecker". Lines in 3D can easily be given by listing the coordinates of two distinct points, for a total of six numbers. Or, they can be given as the coordinates of two distinct planes, eight numbers. What's wrong with these? Nothing; but we can do better. Pluecker coordinates are, in a sense, halfway between these extremes, and can trivially generate either. Neither extreme is as efficient as Pluecker coordinates for computations.

When Pluecker coordinates generalize to Grassmann coordinates, as laid out beautifully in [Hodge], Chapter VII, the determinant definition is clearly the one to use. But 3D lines can use a simpler definition. Take two distinct points on a line, say

```
P = (Px,Py,Pz)
Q = (Qx,Qy,Qz)
```

Think of these as vectors from the origin, if you like. The Pluecker coordinates for the line are essentially

```
U = P - Q
V = P x Q
```

Except for a scale factor, which we ignore, U and V do not depend on the specific P and Q! Cross products are perpendicular to their factors, so we always have $U.V = 0$. In [Stolfi] lines have orientation, so are the same only if their Pluecker coordinates are related by a positive scale factor.

As determinants of homogeneous coordinates, begin with the 4x2 matrix

```
[ Px  Qx ] row x
[ Py  Qy ] row y
[ Pz  Qz ] row z
[  1   1 ] row w
```

Define Pluecker coordinate Gij as the determinant of rows i and j, in that order. Notice that Giw = Pi - Qi, which is Ui. Now let (i,j,k) be a cyclic permutation of (x,y,z), namely (x,y,z) or (y,z,x) or (z,x,y), and notice that Gij = Vk. Determinants are anti-symmetric in the rows, so Gij = -Gji. Thus all possible Pluecker line coordinates are either zero (if i=j) or components of U or V, perhaps with a sign change. Taking the w component of a vector as 0, the determinant form will operate just as well on a point P and vector U as on two points. We can also begin with a 2x4 matrix whose rows are the coefficients of homogeneous plane equations, E.P=0, from which come dual coordinates G'ij. Now if (h,i,j,k) is an even permutation of (x,y,z,w), then Ghi = G'jk. (Just swap U and V!)

Got Pluecker, want points? No problem. At least one component of U is non-zero, say Ui, which is Giw. Create homogeneous points Pj = Gjw + Gij, and Qj = Gij. (Don't expect the P and Q that we started with, and don't expect w=1.) Want plane equations? Let (i,j,k,w) be an even permutation of (x,y,z,w), so G'jk = Giw. Then create Eh = G'hk, and Fh = G'jh.

Example: Begin with P = (2,4,8) and Q = (2,3,5). Then U = (0,1,3) and V = (-4,6,-2). The direct determinant forms are Gxw=0, Gyw=1, Gzw=3, Gyz=-4, Gzx=6, Gxy=-2, and the dual forms are G'yz=0, G'zx=1, G'xy=3, G'xw=-4, G'yw=6, G'zw=-2. Take Uz = Gzw = G'xy = 3 as a suitable non-zero element. Then two planes meeting in the line are

```
(G'xy  G'yy  G'zy  G'wy).P = 0
(G'xx  G'xy  G'xz  G'xw).P = 0
```

That is, a point P is on the line if it satisfies both these equations:

```
3 Px + 0 Py + 0 Pz - 6 Pw = 0
0 Px + 3 Py - 1 Pz - 4 Pw = 0
```

We can also easily determine if two lines meet, or if not, how they pass. If U1 and V1 are the coordinates of line 1, U2 and V2, of line 2, we look at the sign of U1.V2 + V1.U2. If it's zero, they meet. The determinant form reveals even permutations of (x,y,z,w): G1xw G2yz + G1yw G2zx + G1zw G2xy + G1yz G2xw + G1zx p2yw + G1xy p2zw

Two oriented lines L1 and L2 can interact in three different ways: L1 might intersect L2, L1 might go clockwise around L2, or L1 might go counterclockwise around L2. Here are some examples:

```
     | L2            | L2            | L2
L1   |          L1   |          L1   |
-----+----->    ----------->    -----|----->
     |               |               |
     V               V               V
 intersect    counterclockwise   clockwise
     | L2            | L2            | L2
L1   |          L1   |          L1   |
<----+-----     <----|------    <-----------
     |               |               |
     V               V               V
```

The first and second rows are just different views of the same lines, once from the "front" and once from the "back." Here's what they might look like if you look straight down line L2 (shown here as a dot).

```
L1                              ---------->
-----o---->     L1   o          L1   o
                ---------->
 intersect    counterclockwise    clockwise
```

The Pluecker coordinates of L1 and L2 give you a quick way to test which of the three it is.

```
cw:   U1.V2 + V1.U2 < 0
ccw:  U1.V2 + V1.U2 > 0
thru: U1.V2 + V1.U2 = 0
```

So why is this useful? Suppose you want to test if a ray intersects a triangle in 3-space. One way to do this is to represent the ray and the edges of the triangle with Pluecker coordinates. The ray hits the triangle if and only if it hits one of the triangle's edges, or it's "clockwise" from all three edges, or it's "counterclockwise" from all three edges. For example...

```
  o  _
  | |\                 ...in this picture, the ray
  |   \                is oriented counterclockwise
------ \ -->           from all three edges, so it
  |     \              must intersect the triangle.
  v      \
  o-----> o
```

Using Pluecker coordinates, ray shooting tests like this take only a few lines of code.

Grassmann coordinates allow similar methods to be used for points, lines, planes, and so on, and in a space of any dimension. Consult [Hodge] for a thorough discussion of the theory, [Stolfi] for a little theory with a concise implementation for low dimensions (see Subj. 0.04), and this article for different but equally lucid description: J. Erickson, Pluecker coordinates, Ray Tracing News 10(3) 1997, *http://www.acm.org/tog/resources/RTNews/html/rtnv10n3.html* .

8 7. Contributors

8.1 : How can you contribute to this FAQ?

Send email to orourke@cs.smith.edu with your suggestions, possible topics, corrections, or pointers to information.

8.2 : Contributors. Who made this all possible.

Jens Alfke *jens_alfke@powertalk.apple.com* Nina Amenta *amenta@cs.utexas.edu* Leen Ammeraal consunet.nl> Scott Anguish *sanguish@digifix.com* Ian Ashdown *byheart@direct.ca*

Barak *pwbarak@facstaff.wisc.edu* Brad Barber geom.umn.edu, barber@tiac.net> James Beech *jab016@students.stir.ac.uk* David Bouman orthopaedic-surgery.oxford.ac.uk> Paul Bourke *pdb@mhri.edu.su* Andrew Bromage *bromage@mundil.cs.mu.OZ.AU* Brent Burley *brent@aimla.com* R. Kevin Burton *engr@waun.tdsnet.com* Gene Caldwell *optic@halcyon.com* Ken Clarkson *clarkson@research.att.com* Martin Dillon *martind@mv.pi.csiro.au* Thomas Djafari *frogger@micronet.fr* Dave Eberly magic-software.com> John Eickemeyer *johne@iti.gov.sg* John E (Edward) Ellis *je_ellis@ccmail.pnl.gov* Jeff Erickson *jeffe@cs.duke.edu* Ata Etemadi team17.com> Robert W. Fuentes *fuentesrw@lmtas.lmco.com* David N. Fogel *fogel@geog.ucsb.edu* Arne K. Frick *africk@tomsawyer.com* Olexandr Frantchuk my-deja.com> Komei Fukuda *fukuda@ifor.math.ethz.ch* Normand Grégoire *normand@contact.net* Eric Haines *erich@eye.com* Jeff Hameluck *hameluck@wetafx.co.nz* Sandy Harris *sandy@storm.ca* Luiz Henrique de Figueiredo *lhf@visgraf.impa.br* Steve Hollasch *hollasch@kgc.com* Bill Jones *jones@cs.usask.ca* Richard Kinch *kinch@truetex.com* Craig Kolb *cek@Princeton.EDU* Piyush Kumar *piyush@acm.org* Steve Lamont *spl@szechuan.ucsd.edu* Erik Larsen *elarsen@parainc.com* Jon Leech *leech@cs.unc.edu* Michael V. Leonov *mvl@rocketmail.com* Sum Lin *slin@esri.com* Alan J. Livingston *ajl@chyron.com* Sebastien Loisel *zed@sgi.com* Fritz Lott *fritz@riverside.MR.Net* Marc Christopher Martin *mcm@post5.tele.dk* John McNamara fusilla.abanet.it> Samuel Murphy *sammy@icarus.smds.com* Alan Murta *amurta@cs.man.ac.uk* S. N. Muthukrishnan *muthu@macsch.com* Aaron Orenstein *aorenste@atitech.com* Joseph O'Rourke *orourke@cs.smith.edu* Samuel S. Paik *paik@mlo.dec.com* Leonidas Palios *palios@cs.uoi.gr* Amitha Perera *perera@cs.rpi.edu* Brian Peters tplg.ie> Lavoie Philippe *lavoie@zeus.genie.uottawa.ca* Christopher Phillips *christopher@rflect.demon.co.uk* Tom Plunket *plunket@eidetic.com* | Rudi Bjxrn Rasmussen b-rasmussen.com> Greg Roelofs *newt@pobox.com* Christian von Roques *roques@pond.sub.org* Dave Seaman *ags@seaman.cc.purdue.edu* Jonathan R. Shewchuk *Jonathan_R_Shewchuk@ux4.sp.cs.cmu.edu* Rainer Michael Schmid fsinfo.cs.uni-sb.de> Klamer Schutte *klamer@ph.tn.tudelft.nl* | Andrzej Serafin *aserafin@post.pl* ZhengYu Shan *shan@pace10.wh.lucent.com* James Sharman *James@exaflop.demon.co.uk* Ken Shoemake *shoemake@graphics.cis.upenn.edu* Jeff Somers *jsomers@tiac.net* Jon Stone *jdstone@ingr.com* Seth Teller *seth@larch.lcs.mit.edu* Yael "YoeL" Touboul ifp.fr> Anson Tsao *ansont@hookup.net* Bob van Manen *bob@square1.nl* Remco Veltkamp *remcov@cs.ruu.nl* Jim Ward *jfw@radix.net* Jason Weiler *weilej@rpi.edu* Karsten Weiss *karsten@addx.stgt.sub.org* Stefan Wolfrum t-mobil.de>

Previous Editors: Jon Stone *jdstone@ingr.com* Anson Tsao *ansont@hookup.net*