# Circles and spheres

**In what follows are various notes and algorithms dealing with circles and spheres.**

## Spheres, equations and terminology

Written by Paul Bourke
April 1992

OpenGL/GLUT source code demonstrating the Great Circle

**Definition**

The most basic definition of the surface of a sphere is "the set of points an equal distance (called the radius) from a single point called the center". Or as a function of 3 space coordinates (x,y,z), all the points satisfying the following lie on a sphere of radius r centered at the origin

$$x^2 + y^2 + z^2 = r^2$$

For a sphere centered at a point $(x_o,y_o,z_o)$ the equation is simply

$$(x - x_o)^2 + (y - y_o)^2 + (z - z_o)^2 = r^2$$

If the expression on the left is less than $r^2$ then the point (x,y,z) is on the interior of the sphere, if greater than $r^2$ it is on the exterior of the sphere.

A sphere may be defined parametrically in terms of (u,v)

$$x = x_o + r \cos(phi) \cos(theta)$$
$$y = y_o + r \cos(phi) \sin(theta)$$
$$z = z_o + r \sin(phi)$$

Where 0 <= theta < 2 pi, and -pi/2 <= phi <= pi/2. The convention in common usage is for lines of constant theta to run from one pole (phi = -pi/2 for the south pole) to the other pole (phi = pi/2 for the north pole) and are usually referred to as lines of longitude. Lines of constant phi are often referred to as lines of latitude, for example the equator is at phi = 0.

**Lines through a sphere**

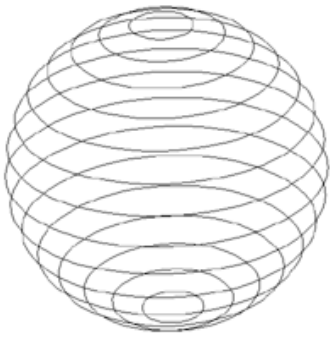
A line can intersect a sphere at one point in which case it is called a tangent. It can not intersect the sphere at all or it can intersect the sphere at two points, the entry and exit points. For the mathematics for the intersection point(s) of a line (or line segment) and a sphere see this.

**Antipodal points**

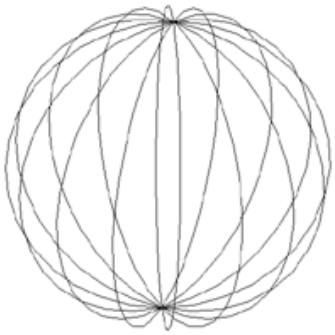
A line that passes through the center of a sphere has two intersection points, these are called antipodal points.

**Planes through a sphere**

A plane can intersect a sphere at one point in which case it is called a tangent plane. Otherwise if a plane intersects a sphere the "cut" is a circle. Lines of latitude are examples of planes that intersect the Earth sphere.

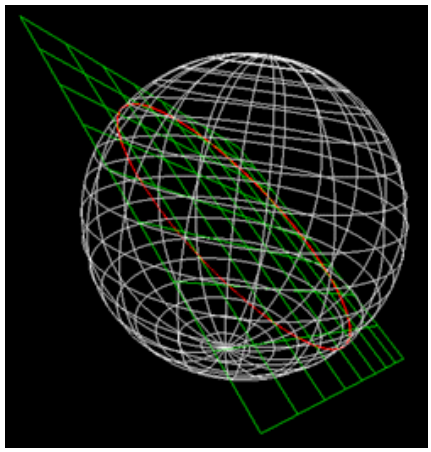


Lines of latitude



Lines of longitude (Meridians)

**Great Circles**



A great circle is the intersection a plane and a sphere where the plane also passes through the center of the sphere. Lines of longitude and the equator of the Earth are examples of great circles. Two points on a sphere that are not antipodal define a unique great circle, it traces the shortest path between the two points. If the points are antipodal there are an infinite number of great circles that pass through them, for example, the antipodal points of the north and south pole of Earth (there are of course infinitely many others).

Great circles define geodesics for a sphere. (A geodesic is the closest path between two points on any surface).

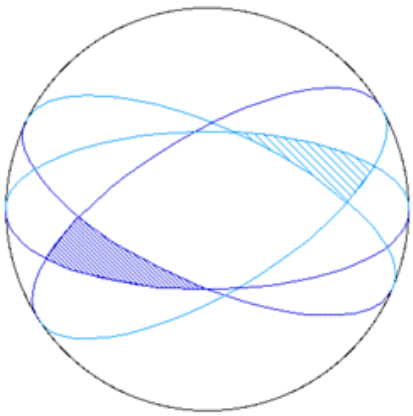
**Lune**

A lune is the area between two great circles who share antipodal points. If the angle between the planes defining the great circle is A, then the area of a lune on a sphere of radius r is

$$\text{area} = 2\ A\ r^2$$

**Triangles**

A triangle on a sphere is defined as the intersecting area of three great circles. Unlike a plane where the interior angles of a triangle sum to pi radians (180 degrees), on a sphere the interior angles sum to more than pi. As the sphere becomes large compared to the triangle then the the sum of the internal angles approach pi.



The area of a spherical triangle with internal angles A,B,C is simply

$$\text{area} = r^2\ (\ A + B + C - \text{pi}\ )$$

In terms of the lengths of the sides of the spherical triangle a,b,c then

$$\text{area} = 4 \arctan [\ \text{sqrt}(\ \tan(m/2)\ \tan((m-a)/2)\ \tan((m-b)/2)\ \tan((m-c)/2)\ )\ ]$$
$$\text{where } m = (\ a+b+c\ ) / 2$$

A similar result for a four sided polygon on the surface of a sphere is

$$\text{area} = r^2\ (\ A + B + C + D - 2\ \text{pi}\ )$$

**Ellipsoid**

An ellipsoid squashed along each (x,y,z) axis by a,b,c is defined as

$$[(x - x_o) / a]^2 + [(y - y_o) / b]^2 + [(z - z_o) / c]^2 = r^2$$

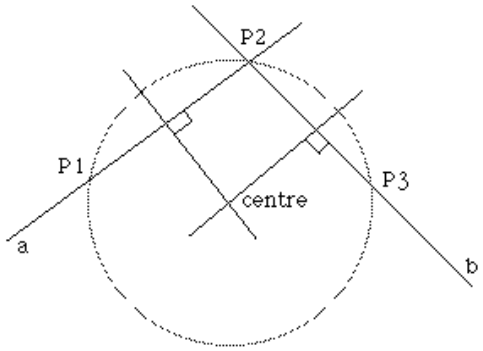
Or parametrically

$$x = x_o + a\ r\ \cos(\text{theta})\ \cos(\text{phi})$$
$$y = y_o + b\ r\ \cos(\text{theta})\ \sin(\text{phi})$$
$$z = z_o + c\ r\ \sin(\text{theta})$$

# Equation of a Circle from 3 Points (2 dimensions)

Written by Paul Bourke
January 1990

This note describes a technique for determining the attributes of a circle (centre and radius) given three points $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$ on a plane.




**Calculating Centre**

Two lines can be formed through 2 pairs of the three points, the first passes through the first two points $\mathbf{P_1}$ and $\mathbf{P_2}$. Line b passes through the next two points $\mathbf{P_2}$ and $\mathbf{P_3}$.
The equation of these two lines is

$$y_a = m_a (x - x_1) + y_1 \quad \text{and} \quad y_b = m_b (x - x_2) + y_2$$

where m is the slope of the line given by

$$m_a = \frac{y_2 - y_1}{x_2 - x1} \quad \text{and} \quad m_b = \frac{y_3 - y_2}{x_3 - x_2}$$

The centre of the circle is the intersection of the two lines perpendicular to and passing through the midpoints of the lines $\mathbf{P_1P_2}$ and $\mathbf{P_2}$ $\mathbf{P_3}$. The perpendicular of a line with slope m has slope -1/m, thus equations of the lines perpendicular to lines a and b and passing through the midpoints of $\mathbf{P_1P_2}$ and $\mathbf{P_2P_3}$ are

$$y'_a = -\frac{1}{m_a}\left(x - \frac{x_1 + x_2}{2}\right) + \frac{y_1 + y_2}{2}$$
$$y'_b = -\frac{1}{m_b}\left(x - \frac{x_2 + x_3}{2}\right) + \frac{y_2 + y_3}{2}$$

These two lines intersect at the centre, solving for x gives

$$x = \frac{m_a m_b (y_1 - y_3) + m_b (x_1 + x_2) - m_a (x_2 + x_3)}{2(m_b - m_a)}$$

Calculate the y value of the centre by substituting the x value into one of the equations of the perpendiculars. Alternatively one can also rearrange the equations of the perpendiculars and solve for y.

**Radius**

The radius is easy, for example the point $\mathbf{P_1}$ lies on the circle and we know the centre....

**Notes:**

- The denominator (mb - ma) is only zero when the lines are parallel in which case they must be coincident and thus no circle results.
- If either line is vertical then the corresponding slope is infinite. This can be solved by simply rearranging the order of the points so that vertical lines do not occur.

**Source Code**

C++ code implemented as MFC (MS Foundation Class) supplied by Jae Hun Ryu. Circle.cpp, Circle.h.

# Equation of a Sphere from 4 Points on the Surface

Written by Paul Bourke
June 2002

Given 4 points in 3 dimensional space [ $(x_1,y_1,z_1)$ $(x_2,y_2,z_2)$ $(x_3,y_3,z_3)$ $(x_4,y_4,z_4)$ ] the equation of the sphere with those points on the surface is found by solving the following determinant.

$$\begin{vmatrix} x^2+y^2+z^2 & x & y & z & 1 \\ x_1^2+y_1^2+z_1^2 & x_1 & y_1 & z_1 & 1 \\ x_2^2+y_2^2+z_2^2 & x_2 & y_2 & z_2 & 1 \\ x_3^2+y_3^2+z_3^2 & x_3 & y_3 & z_3 & 1 \\ x_4^2+y_4^2+z_4^2 & x_4 & y_4 & z_4 & 1 \end{vmatrix} = 0$$

There are conditions on the 4 points, they are listed below and correspond to the determinant above being undefined (no solutions, multiple solutions, or infinite solutions).

- No three combinations of the 4 points can be collinear.
- All 4 points cannot lie on the same plane (coplanar).

If the determinant is found using the expansion by minors using the top row then the equation of the sphere can be written as follows.

$$(x^2+y^2+z^2) \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ & & & 1 \end{vmatrix} - x \begin{vmatrix} x_1^2+y_1^2+z_1^2 & y_1 & z_1 & 1 \\ x_2^2+y_2^2+z_2^2 & y_2 & z_2 & 1 \end{vmatrix}$$

$$\begin{vmatrix} x_3 & y_3 & z_3 & \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \quad \begin{vmatrix} x_3^2 + y_3^2 + z_3^2 & y_3 & z_3 & 1 \\ x_4^2 + y_4^2 + z_4^2 & y_4 & z_4 & 1 \end{vmatrix}$$

$$+ y \begin{vmatrix} x_1^2 + y_1^2 + z_1^2 & x_1 & z_1 & 1 \\ x_2^2 + y_2^2 + z_2^2 & x_2 & z_2 & 1 \\ x_3^2 + y_3^2 + z_3^2 & x_3 & z_3 & 1 \\ x_4^2 + y_4^2 + z_4^2 & x_4 & z_4 & 1 \end{vmatrix} - z \begin{vmatrix} x_1^2 + y_1^2 + z_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 + z_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 + z_3^2 & x_3 & y_3 & 1 \\ x_4^2 + y_4^2 + z_4^2 & x_4 & y_4 & 1 \end{vmatrix} + \begin{vmatrix} x_1^2 + y_1^2 + z_1^2 & x_1 & y_1 & z_1 \\ x_2^2 + y_2^2 + z_2^2 & x_2 & y_2 & z_2 \\ x_3^2 + y_3^2 + z_3^2 & x_3 & y_3 & z_3 \\ x_4^2 + y_4^2 + z_4^2 & x_4 & y_4 & z_4 \end{vmatrix} = 0$$

Or more simply in term of the minors $M_{1j}$

$$(x^2 + y^2 + z^2)\, M_{11} - x\, M_{12} + y\, M_{13} - z\, M_{14} + M_{15} = 0$$

The general equation of a sphere with radius r centered at $(x_0,y_0,z_0)$ is

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$$

Equating the terms from these two equations allows one to solve for the center and radius of the sphere, namely:

$$x_0 = 0.5\, M_{12} / M_{11}$$

$$y_0 = -\, 0.5\, M_{13} / M_{11}$$

$$z_0 = 0.5\, M_{14} / M_{11}$$

$$r^2 = x_0^2 + y_0^2 + z_0^2 - M_{15} / M_{11}$$

Note that these can't be solved for $M_{11}$ equal to zero. This corresponds to no quadratic terms ($x^2$, $y^2$, $z^2$) in which case we aren't dealing with a sphere and the points are either coplanar or three are collinear.

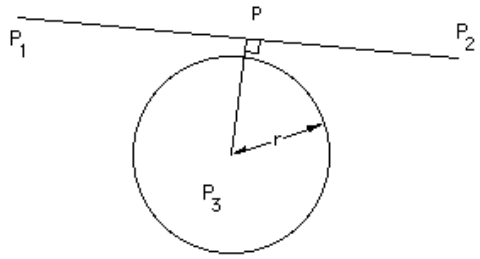
**C source code**

This piece of simple C code tests the solution as described above. It creates a known sphere (center and radius) and creates 4 random points on that sphere. It then proceeds to find the original center and radius using those four random points.

# Intersection of a Line and a Sphere (or circle)

Written by Paul Bourke

November 1992




Points **P** (x,y) on a line defined by two points $\mathbf{P}_1$ ($x_1$,$y_1$,$z_1$) and $\mathbf{P}_2$ ($x_2$,$y_2$,$z_2$) is described by

$$\mathbf{P} = \mathbf{P}_1 + u\,(\mathbf{P}_2 - \mathbf{P}_1)$$

or in each coordinate

$$x = x_1 + u\,(x_2 - x_1)$$
$$y = y_1 + u\,(y_2 - y_1)$$
$$z = z_1 + u\,(z_2 - z_1)$$

A sphere centered at $\mathbf{P}_3$ ($x_3$,$y_3$,$z_3$) with radius r is described by

$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r^2$$

Substituting the equation of the line into the sphere gives a quadratic equation of the form

$$a\,u^2 + b\,u + c = 0$$

where:

$$a = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2$$

$$b = 2[\,(x_2 - x_1)\,(x_1 - x_3) + (y_2 - y_1)\,(y_1 - y_3) + (z_2 - z_1)\,(z_1 - z_3)\,]$$

$$c = x_3^2 + y_3^2 + z_3^2 + x_1^2 + y_1^2 + z_1^2 - 2[x_3\,x_1 + y_3\,y_1 + z_3\,z_1] - r^2$$

The solutions to this quadratic are described by

$$\frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2a}$$

The exact behaviour is determined by the expression within the square root

$$b * b - 4 * a * c$$

- If this is less than 0 then the line does not intersect the sphere.
- If it equals 0 then the line is a tangent to the sphere intersecting it at one point, namely at u = -b/2a.
- If it is greater then 0 the line intersects the sphere at two points.

To apply this to two dimensions, that is, the intersection of a line and a circle simply remove the z component from the above mathematics.

## Line Segment

For a line segment between $\mathbf{P}_1$ and $\mathbf{P}_2$ there are 5 cases to consider.

- Line segment doesn't intersect and on outside of sphere, in which case both values of u will either be less than 0 or greater than 1.
- Line segment doesn't intersect and is inside sphere, in which case one value of u will be negative and the other greater than 1.
- Line segment intersects at one point, in which case one value of u will be between 0 and 1 and the other not.
- Line segment intersects at two points, in which case both values of u will be between 0 and 1.
- Line segment is tangential to the sphere, in which case both values of u will be the same and between 0 and 1.

When dealing with a line segment it may be more efficient to first determine whether the line actually intersects the sphere or circle. This is achieved by noting that the closest point on the line through $\mathbf{P}_1\mathbf{P}_2$ to the point $\mathbf{P}_3$ is along a perpendicular from $\mathbf{P}_3$ to the line. In other words if $\mathbf{P}$ is the closest point on the line then

$$(\mathbf{P}_3 - \mathbf{P}) \text{ dot } (\mathbf{P}_2 - \mathbf{P}_1) = 0$$

Substituting the equation of the line into this

$$[\mathbf{P}_3 - \mathbf{P}_1 - u(\mathbf{P}_2 - \mathbf{P}_1)] \text{ dot } (\mathbf{P}_2 - \mathbf{P}_1) = 0$$

Solving the above for u =

$$\frac{(x_3 - x_1)(x_2 - x_1) + (y_3 - y_1)(y_2 - y_1) + (z_3 - z_1)(z_2 - z_1)}{(x_2 - x_1)(x_2 - x_1) + (y_2 - y_1)(y_2 - y_1) + (z_2 - z_1)(z_2 - z_1)}$$

If u is not between 0 and 1 then the closest point is not between $\mathbf{P}_1$ and $\mathbf{P}_2$ Given u, the intersection point can be found, it must also be less than the radius r. If these two tests succeed then the earlier calculation of the actual intersection point can be applied.

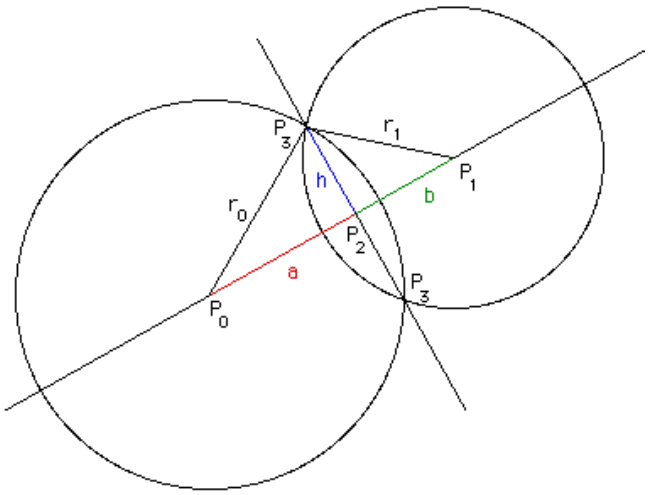
**Source code**

C code example by author.
Source code example by Iebele Abel.
Sphere/ellipse and line intersection code for Visual Basic by Adrian DeAngelis.
Python (3.2) version by Campbell Barton.
LISP version for AutoCAD (and Intellicad) by Andrew Bennett intC2.lsp and intC2_app.lsp.
VBA implementation by Giuseppe Iaria.
VBA/VB6 implementation by Thomas Ludewig.

# Intersection of two circles

Written by Paul Bourke
April 1997

The following note describes how to find the intersection point(s) between two circles on a plane, the following notation is used. The aim is to find the two points $P_3 = (x_3, y_3)$ if they exist.




First calculate the distance d between the center of the circles. $d = ||P_1 - P_0||$.

- If $d > r_0 + r_1$ then there are no solutions, the circles are separate.
- If $d < |r_0 - r_1|$ then there are no solutions because one circle is contained within the other.
- If $d = 0$ and $r_0 = r_1$ then the circles are coincident and there are an infinite number of solutions.

Considering the two triangles $P_0P_2P_3$ and $P_1P_2P_3$ we can write

$$a^2 + h^2 = r_0^2 \text{ and } b^2 + h^2 = r_1^2$$

Using $d = a + b$ we can solve for a,

$$a = (r_0^2 - r_1^2 + d^2 ) / (2\ d)$$

It can be readily shown that this reduces to $r_0$ when the two circles touch at one point, ie: $d = r_0 \pm r_1$

Solve for h by substituting a into the first equation, $h^2 = r_0^2 - a^2$

So

$$P_2 = P_0 + a\ ( P_1 - P_0 )\ /\ d$$

And finally, $P_3 = (x_3,y_3)$ in terms of $P_0 = (x_0,y_0)$, $P_1 = (x_1,y_1)$ and $P_2 = (x_2,y_2)$, is

$$x_3 = x_2 +\text{-}\ h\ ( y_1 - y_0 )\ /\ d$$

$$y_3 = y_2 \text{-}+\ h\ ( x_1 - x_0 )\ /\ d$$

**Source code contributions**

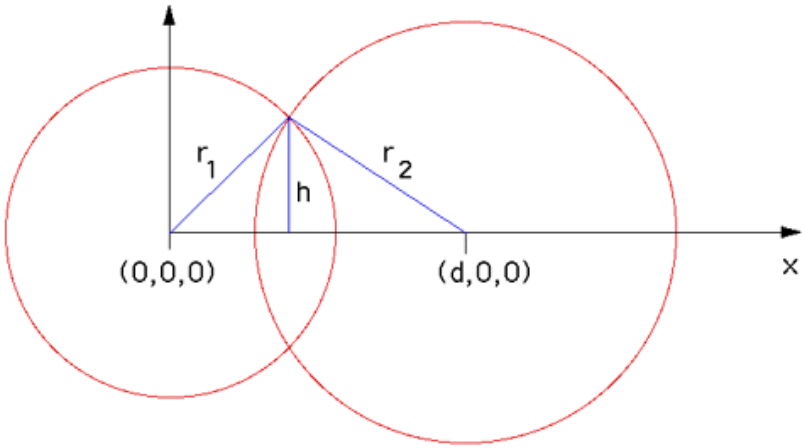
Python version by Matt Woodhead.
C source code example by Tim Voght.
Objective C method by Daniel Quirk.
Contribution from Jonathan Greig.

# Intersection of two spheres

Written by Paul Bourke

November 1995

Consider two spheres on the x axis, one centered at the origin, separated by a distance d, and of radius $r_1$ and $r_2$.




The equations of the two spheres are

$$x^2 + y^2 + z^2 = r_1^2$$

$$(x - d)^2 + y^2 + z^2 = r_2^2$$

Subtracting the first equation from the second, expanding the powers, and solving for x gives

$$x = [ d^2 - r_2^2 + r_1^2] / 2 d$$

The intersection of the two spheres is a circle perpendicular to the x axis, at a position given by x above. Substituting this into the equation of the first sphere gives

$$y^2 + z^2 = [4 d^2 r_1^2 - (d^2 - r_2^2 + r_1^2)^2 ] / 4 d^2$$

You might recognise this is the equation of a circle with radius h

where

$$h^2 = [4 d^2 r_1^2 - (d^2 - r_2^2 + r_1^2)^2 ] / 4 d^2$$

# Distributing Points on a Sphere

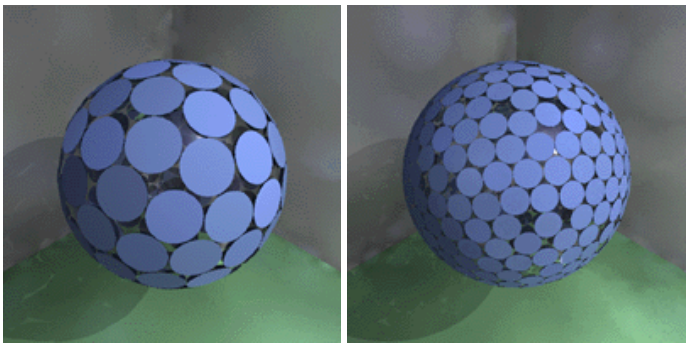
Written by Paul Bourke

June 1996

The following describes two (inefficient) methods of evenly distributing points on a sphere. They do however allow for an arbitrary number of points to be distributed unlike many other algorithms which only work for a restricted set of points.

The first approach is to randomly distribute the required number of points on a sphere of the desired radius. The iteration involves finding the closest two points and then moving them apart slightly.

The following shows the results for 100 and 400 points, the disks have a radius of the minimum distance.

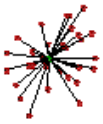


C source code: source1.c

**Physical method**

See Particle Systems for more details on modelling with particle systems.

A more "fun" method is to use a physical particle method. For example we can randomly distribute point particles in 3D space and join each particle to a central fixed particle (intended center of the sphere) with springs with the same rest length.



If we place the same electric charge on each particle (except perhaps the particle in the center) then each particle will repel every other particle. This system will tend to a stable configuration where each particle is equidistant from the center (due to spring forces) and each particle maximally separated from its closest neighbours (electric repulsive forces). It is important to model this with viscous damping as well as with spring damping to avoid oscillatory motion. An example using 31 particles randomly distributed in a cube is shown in the animation above. A midpoint ODE solver was used to solve the equations of motion, it took only 200 steps to reach a stable (minimum energy) configuration.

**Uniform Distribution**

A simple way to randomly (uniform) distribute points on sphere is called the "hypercube rejection method". To apply this to a unit cube at the origin, choose coordinates (x,y,z) each uniformly distributed on the interval [-1,1]. If the length of this vector is greater than 1 then reject it, otherwise normalise it and use it as a sample.

**Contribution by Jonathan D. Lettvin**

C++ source code: diffuse.cpp

**Contribution by Max Downey**

Java implementation: java.tar.gz

**Contribution by Orion Elenzil**

Orion Elenzil proposes that by choosing uniformly distributed polar coordinates theta (0 <= theta < 360) and phi (0 <= phi <= pi/2) but the using the sqrt(phi) results in points uniformly distributed on the surface of a hemisphere. If the poles lie along the z axis then the position on a unit hemisphere sphere is

x = cos(sqrt(phi)) cos(theta)
y = cos(sqrt(phi)) sin(theta)
z = sin(sqrt(phi))

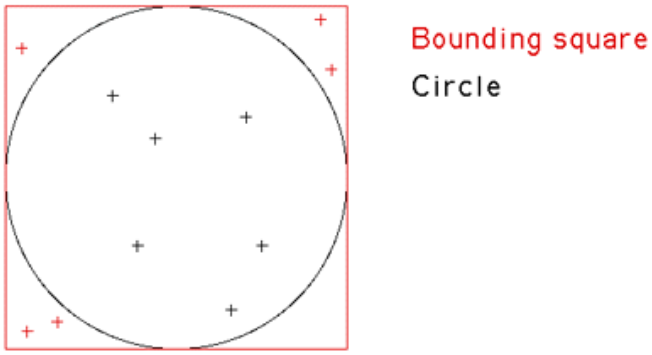
A whole sphere is obtained by simply randomising the sign of z.

# Area of multiple intersecting circles

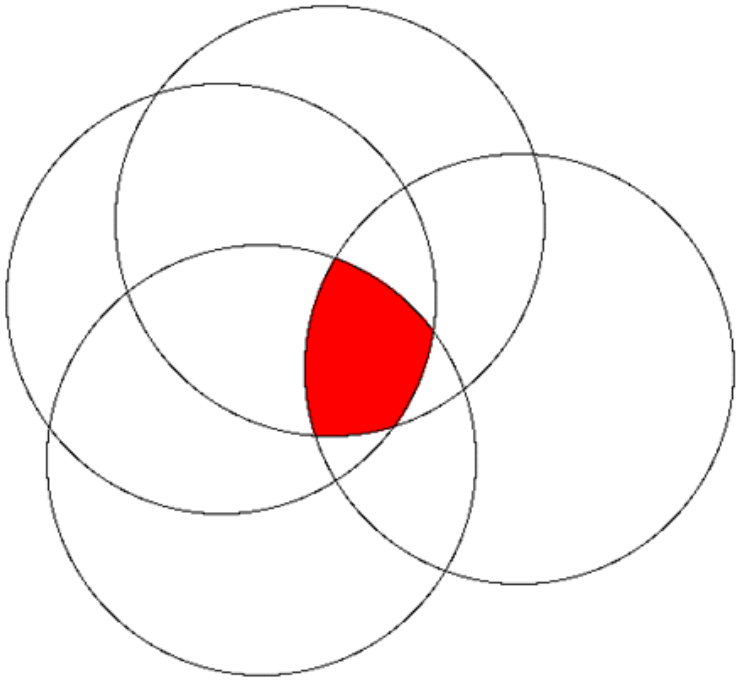
Written by Paul Bourke
January 2000

**Introduction**

The following is a straightforward but good example of a range of techniques called "Monte-Carlo" methods. It will be used here to numerically find the area of intersection of a number of circles on a plane.




Consider a single circle with radius r, the area is pi $r^2$. The minimal square enclosing that circle has sides 2 r and therefore an area of 4 $r^2$. If one was to choose random numbers from a uniform distribution within the bounding rectangle then the ratio of those falling within the circle to the total number will be the ratio of the area of the circle to the rectangle. In other words, countinside/totalcount = pi / 4, this ratio of pi / 4 would be approached closer as the totalcount increases.. This could be used as a way of estimate pi, albeit a very inefficient way!



**Source code**

C source that numerically estimates the intersection area of any number of circles on a plane is given here: area.c. The basic idea is to choose a random point within the bounding square of one of the circles and check to see if the point is within all the other circles. The successful count is scaled by 4 $r^2$ / totalcount to give the area of the intersecting piece.

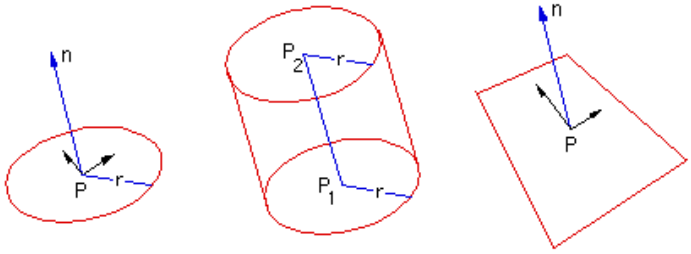
# Creating a plane/disk perpendicular to a line segment

Written by Paul Bourke
February 1997

Contribution by Dan Wills in MEL (Maya Embedded Language): source2.mel.

There are a number of 3D geometric construction techniques that require a coordinate system perpendicular to a line segment, some examples are:

- Creating a disk given its center, radius and normal.
- Forming a cylinder given its two end points and radii at each end.
- Creating a plane coordinate system perpendicular to a line.




A straightforward method will be described which facilitates each of these. The key is deriving a pair of orthonormal vectors on the plane perpendicular to a line segment $P_1$, $P_2$.

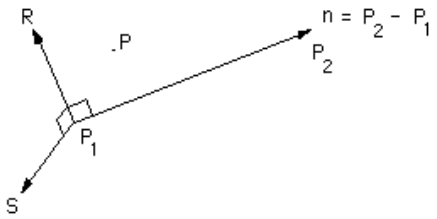
**Procedure**

**1.** Choose any point P randomly which doesn't lie on the line through $P_1$ and $P_2$

**2.** Calculate the vector R as the cross product between the vectors P - $P_1$ and $P_2$ - $P_1$. This vector R is now perpendicular to $P_2$ - $P_1$. (If R is 0 then 1. wasn't satisfied)

**3.** Calculate the vector S as the cross product between the vectors R and $P_2$ - $P_1$. This vector S is now perpendicular to both R and the $P_2$ - $P_1$.

**4.** The unit vectors ||R|| and ||S|| are two orthonormal vectors in the plane perpendicular to $P_2$ - $P_1$.




Points on the plane through $P_1$ and perpendicular to n = $P_2$ - $P_1$ can be found from linear combinations of the unit vectors R and S, for example, a point Q might be

$Q_x = P_{1x} + \text{alpha } R_x + \text{beta } S_x$

$Q_y = P_{1y} + \text{alpha } R_y + \text{beta } S_y$

$Q_z = P_{1z} + \text{alpha } R_z + \text{beta } S_z$

for all alpha and beta.

A disk of radius r, centered at $P_1$, with normal n = $P_2$ - $P_1$ is described as follows

$Q_x = P_{1x} + r \cos(\text{theta}) R_x + r \sin(\text{theta}) S_x$

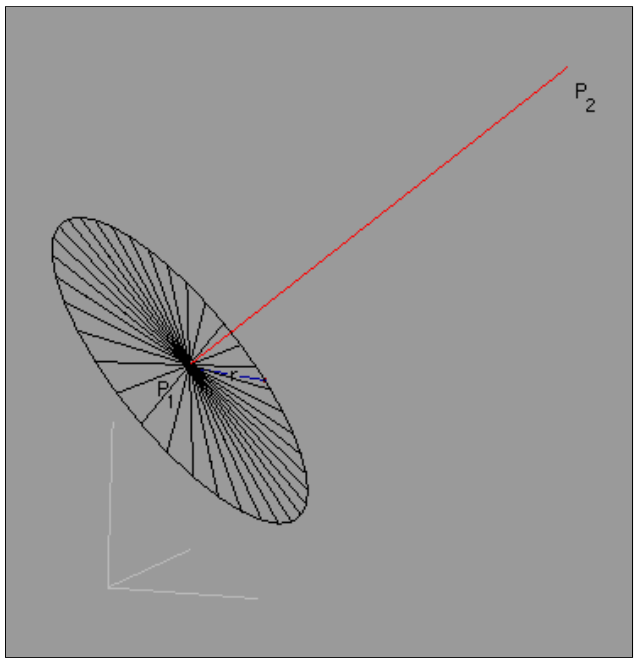
$Q_y = P_{1y} + r \cos(\text{theta}) R_y + r \sin(\text{theta}) S_y$

$Q_z = P_{1z} + r \cos(\text{theta}) R_z + r \sin(\text{theta}) S_z$

for 0 <= theta <= 2 pi

**Example**

The following is a simple example of a disk and the C source stub that generated it.




# Sphere Generation

Written by Paul Bourke
May 1992

## Polar Coordinates

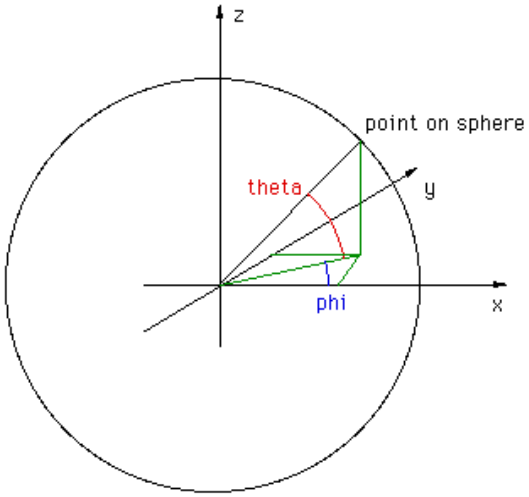
The following illustrate methods for generating a facet approximation to a sphere. The most straightforward method uses polar to Cartesian coordinates, if theta and phi as shown in the diagram below are varied the resulting vector describes points on the surface of a sphere.

The equations of the points on the surface of the sphere are

x = cos(theta) cos(phi)
y = cos(theta) sin(phi)
z = sin(theta)

where

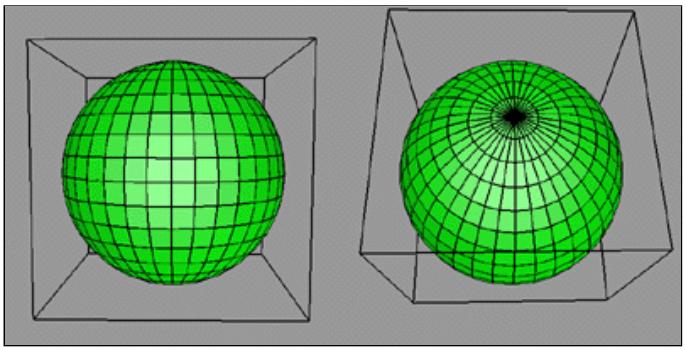
-90 <= theta <= 90
0 <= phi <= 360



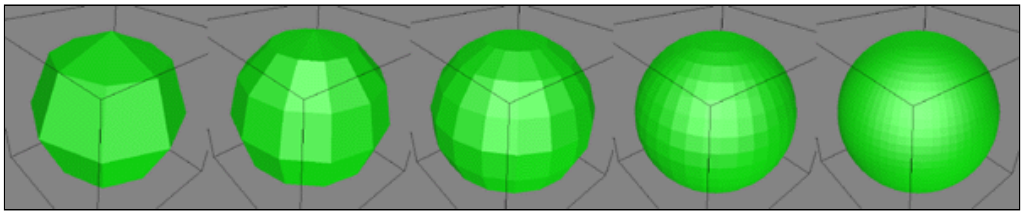

To create a facet approximation, theta and phi are stepped in small angles between their respective bounds. So if we take the angle step size to be dtheta and dphi, the four vertices of any facet correspond to

(theta,phi)
(theta+dtheta,phi)
(theta+dtheta,phi+dphi)
(theta,phi+dphi)

The main drawback with this simple approach is the non uniform resolution (facet size) over the surface of the sphere, in particular, the facets become smaller at the poles.



The sphere can be generated at any resolution, the following shows a progression from 45 degrees through to 5 degree angle increments.

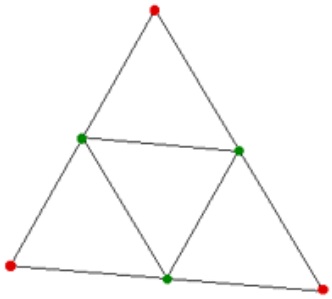


The number of facets being (180 / dtheta) (360 / dphi), the 5 degree example on the right contains almost 2600 facets.

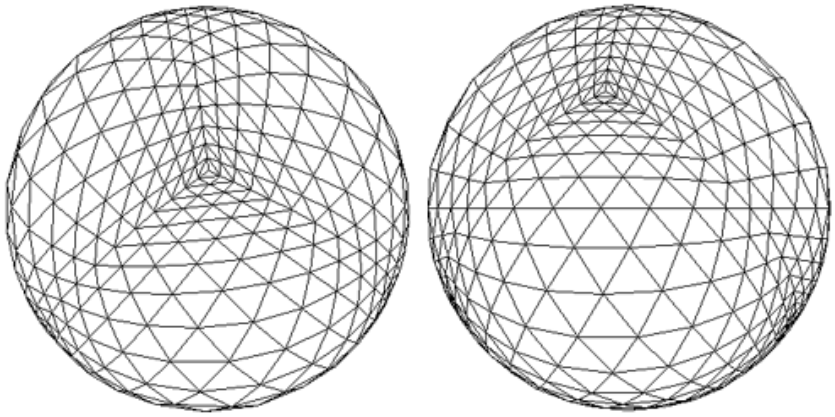
Source code

## Surface refinement

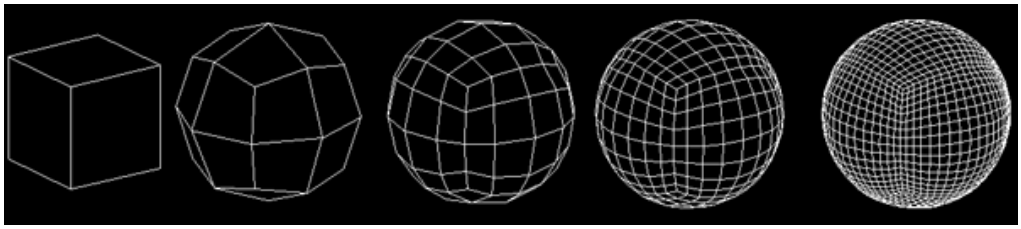
Another method derives a faceted representation of a sphere by starting with a crude approximation and repeatedly bisecting the facets at the same time moving them to the surface of the sphere. The simplest starting form could be a tetrahedron, in the first iteration the 4 facets are split into 4 by bisecting the edges. In each iteration this is repeated, that is, each facet is further split into 4 smaller facets. Either during or at the end of this process (it doesn't matter when) each vertex is moved to the boundary of the sphere by simply normalising the vector and scaling by the desired radius.



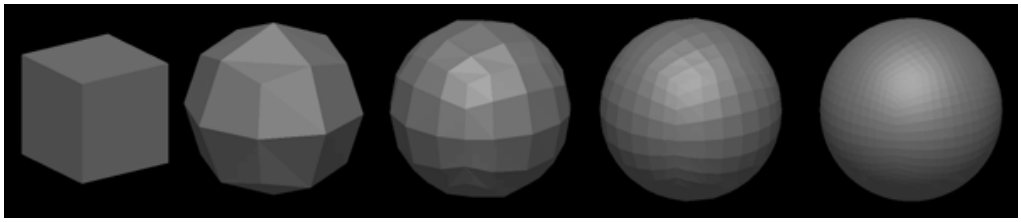
The following illustrates the sphere after 5 iterations, the number of facets increases on each iteration by 4 so this representation has 1024 facets.



Perhaps unexpectedly, all the facets are not the same size, those nearer the vertices of the original tetrahedron are smaller. The above example resulted in a triangular faceted model, if a cube is used as the starting form then a representation with rectangular facets can be derived.

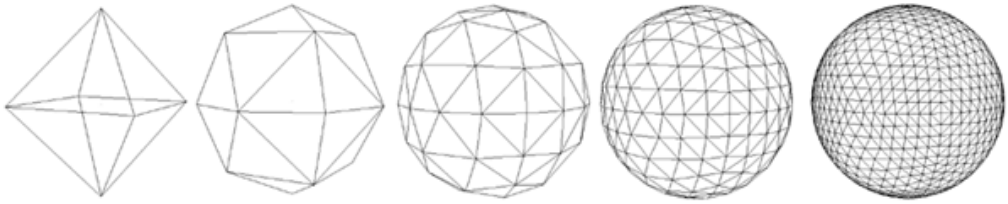


As in the tetrahedron example the facets are split into 4 and thus the number of facets increases by a factor of 4 on each iteration. The representation on the far right consists of 6144 facets.



Source code

The non-uniformity of the facets most disappears if one uses an octahedron as the starting shape. Bisecting the triangular facets results in sphere approximations with 8, 32, 128, 512, 2048, .... facets as the iteration count increases.



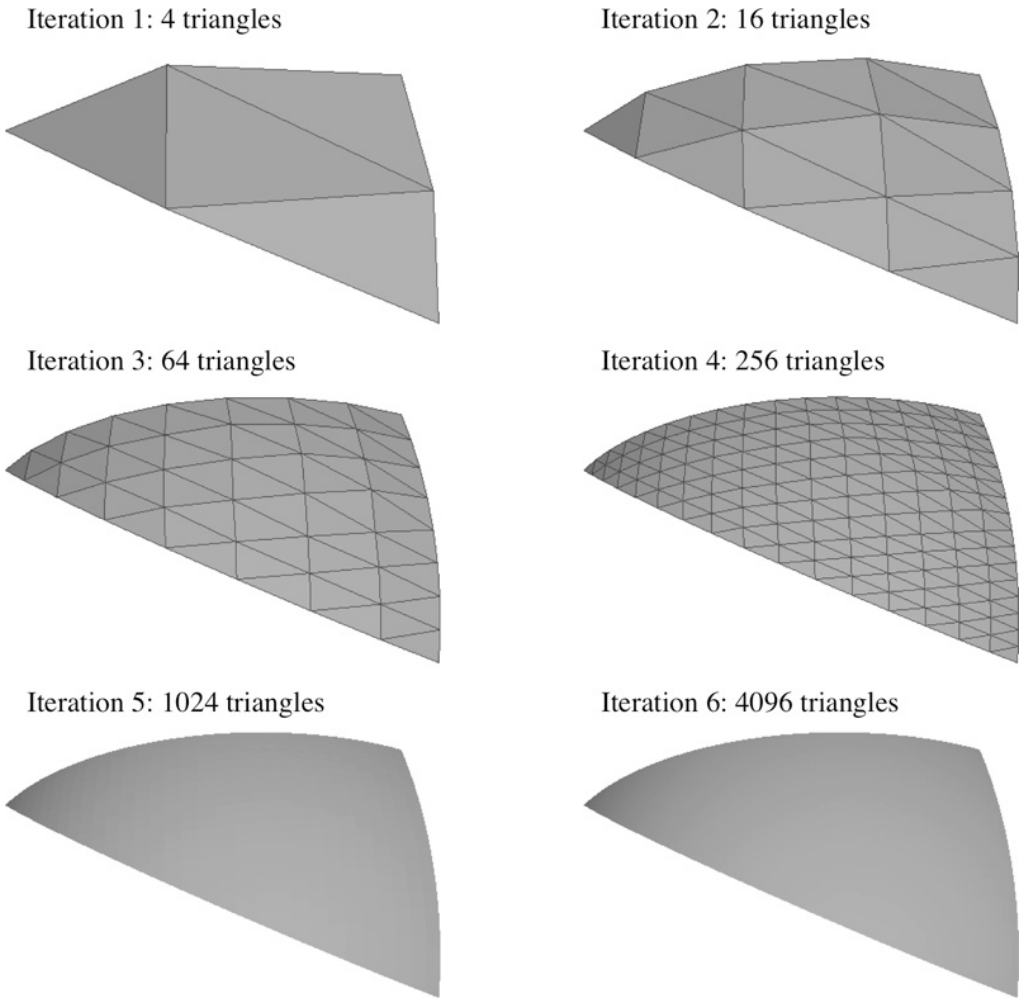
Source code
PovRay example courtesy Louis Bellotto

**Uniform Distribution**

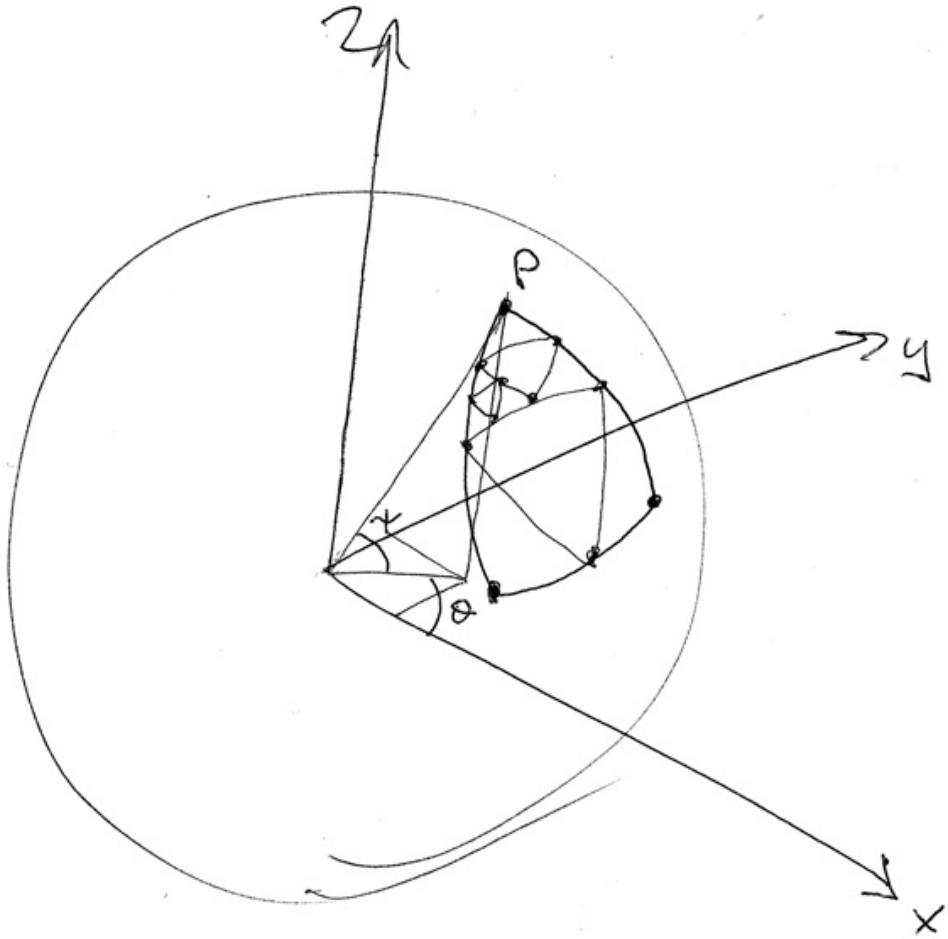
A simple way to randomly (uniform) distribute points on sphere is called the "hypercube rejection method". To apply this to a unit cube at the origin, choose coordinates (x,y,z) each uniformly distributed on the interval [-1,1]. If the length of this vector is greater than 1 then reject it, otherwise normalise it and use it as a sample.

**Spherical triangle**

The same technique can be used to form and represent a spherical triangle, that is, the triangle formed by three points on the surface of a sphere, bordered by three great circle segments.

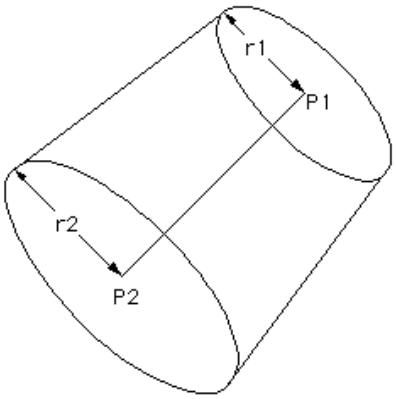



The algorithm and the conventions used in the sample source code provided is illustrated below. The three vertices of the triangle are each defined by two angles, longitude and latitude, on each iteration the number of triangles increases by a factor of 4.



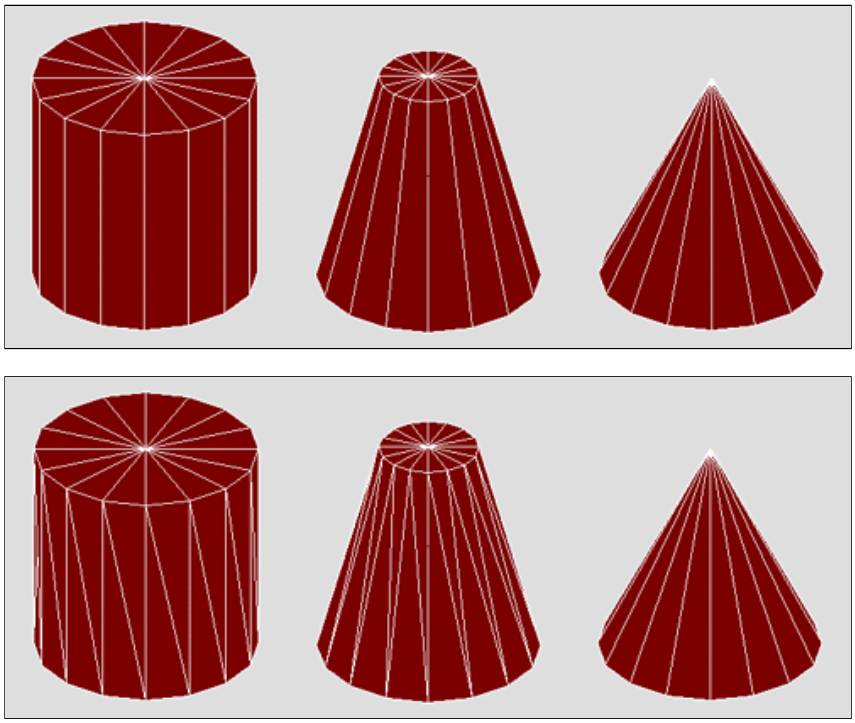

# Facet Approximation to a Cylinder

Written by Paul Bourke
October 1999

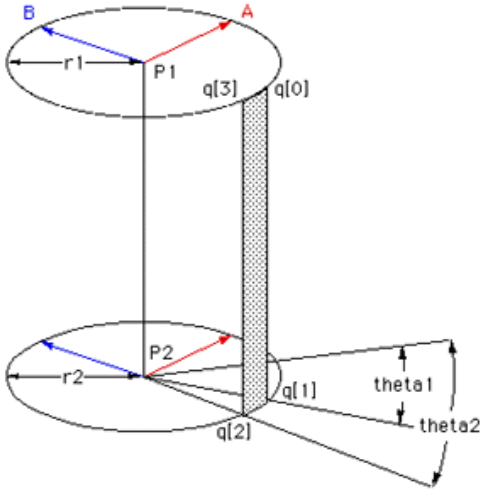



The following describes how to represent an "ideal" cylinder (or cone) by discrete facets. The reasons for wanting to do this mostly stem from environments that don't support a cylinder primitive, for example OpenGL, DXF and STL. A very general definition of a cylinder will be used, it will be defined by two end points and a radius at each end. A traditional cylinder will have the two radii the same, a tapered cylinder will have different radii, a cone will have a zero radius at one end.

The following images show the cylinders with either 4 vertex faces or entirely 3 vertex facets. Note that since the 4 vertex polygons are coplanar, splitting them into two 3 vertex facets doesn't improve the resolution. End caps are normally optional, whether they are needed or not is application dependent.



In order to specify the vertices of the facets making up the cylinder one first needs two vectors that are both perpendicular to the cylinder axis as well as perpendicular to each other. These are shown in red and blue in the figure on the right. There are a number of ways of creating these two vectors, they normally require the formation of any vector that is not collinear with the cylinder axis. The cross product of that vector with the cylinder axis (P2-P1) gives one of the vectors (A say), taking the cross product of this new vector with the axis gives the other vector (B). These two perpendicular vectors are then normalised.




Given the two perpendicular vectors A and B one can create vertices around each rim of the cylinder. So, for a 4 vertex facet the vertices might be given by the following where theta2 - theta1 is some suitably small angle that determines the roughness of the approximation.

q[0] = P1 + r1 * cos(theta1) * A + r1 * sin(theta1) * B
q[1] = P2 + r2 * cos(theta1) * A + r2 * sin(theta1) * B
q[2] = P2 + r2 * cos(theta2) * A + r2 * sin(theta2) * B
q[3] = P1 + r1 * cos(theta2) * A + r1 * sin(theta2) * B

Note P1,P2,A, and B are all vectors in 3 space. r1 and r2 are the radii at the two ends.

Written as some pseudo C code the facets might be created as follows

```
create A and B
for (i=0;i<NFACETS;i++) {
   n = 0;
   theta1 = i * TWOPI / N;
   theta2 = (i+1) * TWOPI / N;
   q[n].x = P1.x + r1 * cos(theta1) * A.x + r1 * sin(theta1) * B.x
   q[n].y = P1.y + r1 * cos(theta1) * A.y + r1 * sin(theta1) * B.y
   q[n].z = P1.z + r1 * cos(theta1) * A.z + r1 * sin(theta1) * B.z
   n++;
   q[n].x = P2.x + r2 * cos(theta1) * A.x + r2 * sin(theta1) * B.x
   q[n].y = P2.y + r2 * cos(theta1) * A.y + r2 * sin(theta1) * B.y
   q[n].z = P2.z + r2 * cos(theta1) * A.z + r2 * sin(theta1) * B.z
   n++;
   if (r2 != 0) {
      q[n].x = P2.x + r2 * cos(theta2) * A.x + r2 * sin(theta2) * B.x
      q[n].y = P2.y + r2 * cos(theta2) * A.y + r2 * sin(theta2) * B.y
      q[n].z = P2.z + r2 * cos(theta2) * A.z + r2 * sin(theta2) * B.z
```

```
            n++;
        }
        if (r1 != 0) {
            q[n].x = P1.x + r1 * cos(theta2) * A.x + r1 * sin(theta2) * B.x
            q[n].y = P1.y + r1 * cos(theta2) * A.y + r1 * sin(theta2) * B.y
            q[n].z = P1.z + r1 * cos(theta2) * A.z + r1 * sin(theta2) * B.z
            n++;
        }
        do something with q[0..n]
    }
```

**Note**

- The algorithm described here will cope perfectly well with negative radii. If one radius is negative and the other positive then the cylinder will cross through at a single point, effectively looking like two end-to-end cones. This does lead to facets that have a twist in them which is not always allowed.
- The end caps are simply formed by first checking the radius at each end, if it is not 0 then additional 3 vertex faces are created with vertices P1, q[0], q[3] and/or P2, q[1], q[2].
- If your application requires only 3 vertex facets then the 4 vertex facets above can be split into q[0], q[1], q[2] and q[0], q[2], q[3].
- Pay attention to any facet orderings requirements of your application. Many packages expect normals to be pointing outwards, the exact ordering of the vertices also depends on whether you are using a left or right handed coordinate system.

C source that creates a cylinder for OpenGL

# Modelling with Spheres

Written by Paul Bourke

August 1991

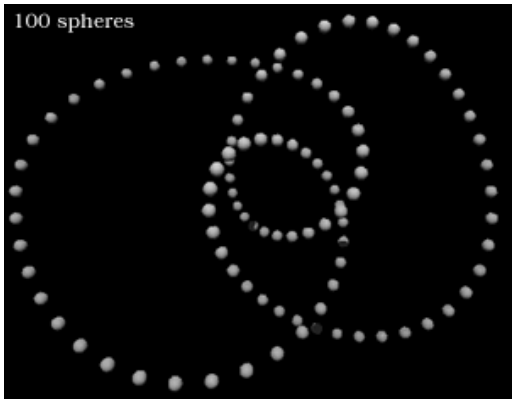
Many computer modelling and visualisation problems lend themselves to placing markers at points in 3 space. It may be that such markers are a natural consequence of the object being studied (for example: chaotic attractors) or it may be that forming other higher level primitives such as tubes or planar facets may be problematic given the description of the object being modelled.

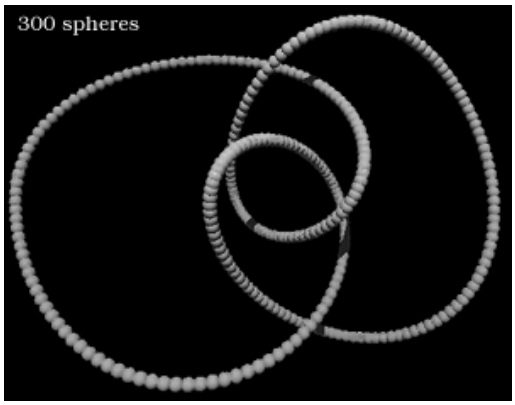
A simple and directionally symmetric marker is the sphere, a point is discounted here, even though it can be considered to be a sphere of zero radius, because most rendering packages do not support such ideal non-real entities. (A ray from a raytracer will never intersect a point which occupies no volume, in the same way, lines can generally not be rendered)

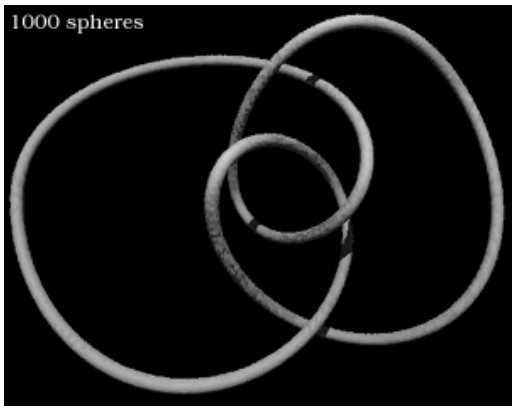
Another reason for wanting to model using spheres as markers is that many rendering packages handle spheres very efficiently. The computationally expensive part of raytracing geometric primitives is testing the intersection of a ray with the primitive. Such a test for a sphere is the most efficient of all primitives, one only needs to determine whether the closest position of the center of the sphere to the ray is less than the radius of the sphere.

## Curve Example

The first example will be modelling a curve in space. The figures below show the same curve represented with an increased number of points, a sphere at each point.

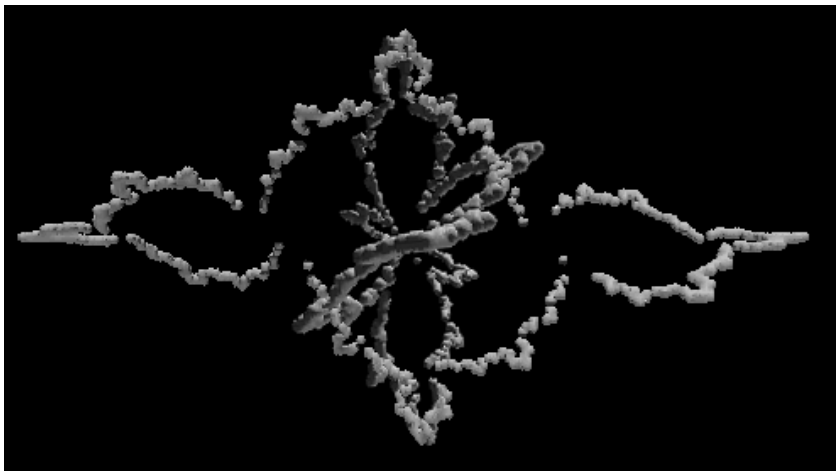


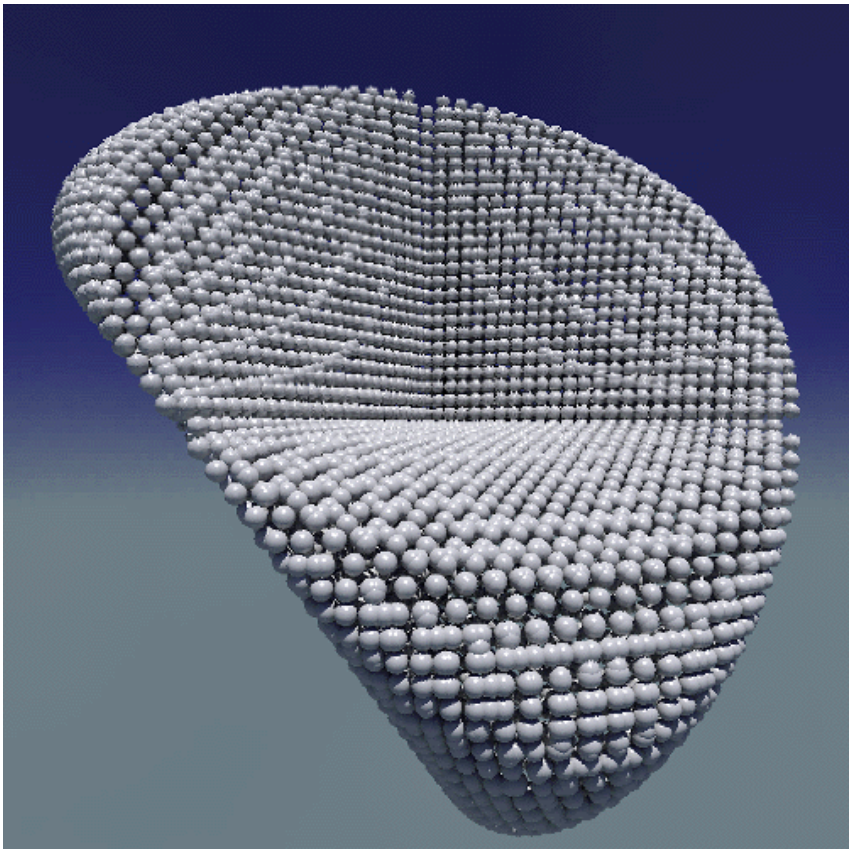

## Chaotic Example

Modelling chaotic attractors is a natural candidate for modelling with spheres because the points are not generated sequentially.



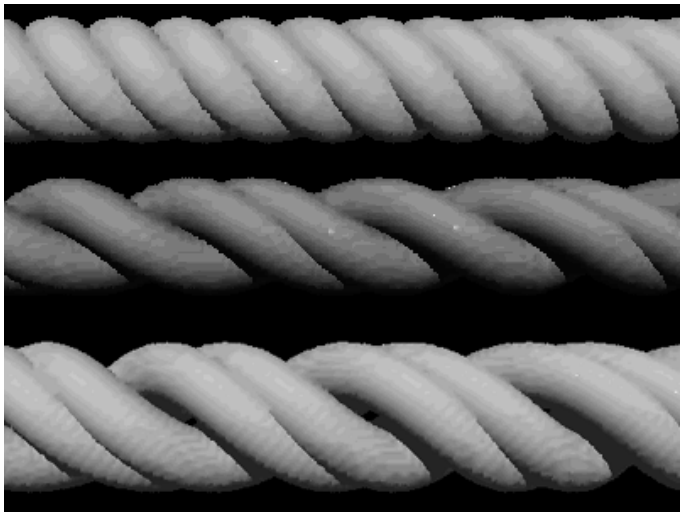
## Surface Example

Surfaces can also be modelled with spheres although this can obviously be very inefficient. The following is an example from a project to visualise the Steiner surface.



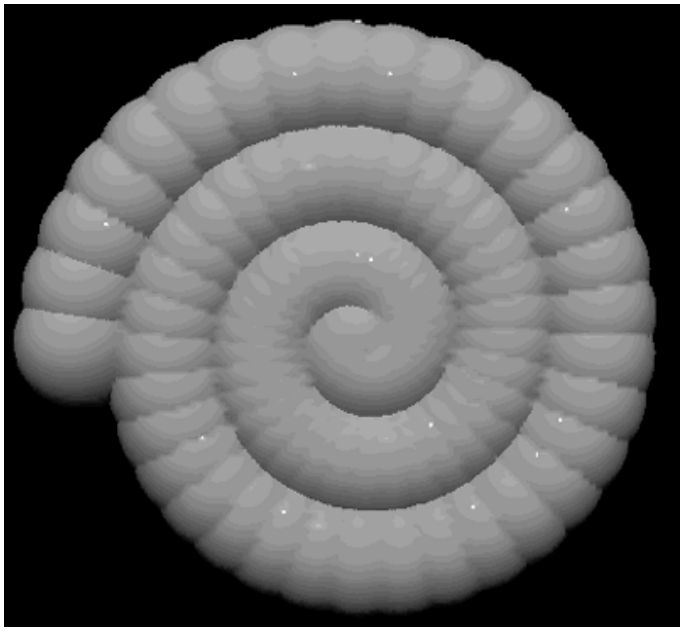
## Modelling Rope

Each strand of the rope is modelled as a series of spheres, each tracing a sinusoidal route through space.

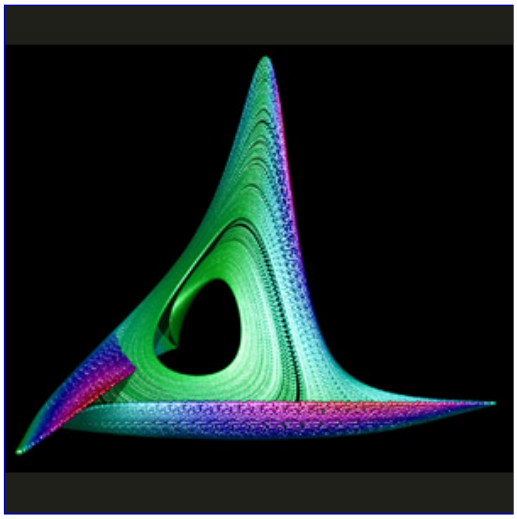


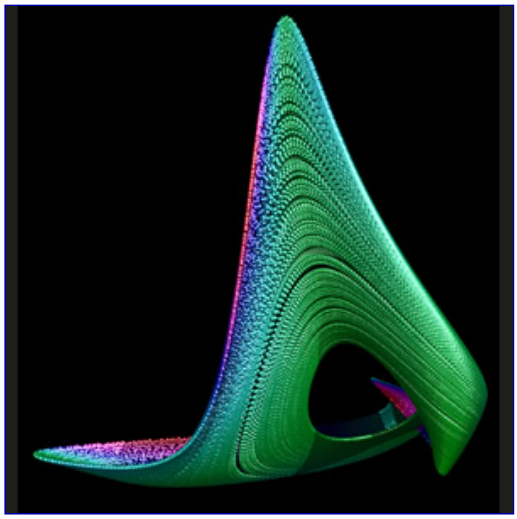
## Sea Shells

Some biological forms lend themselves naturally to being modelled with spherical building blocks as it adds an existing surface texture. Some sea shells for example have a rippled effect



**Representing attractors**





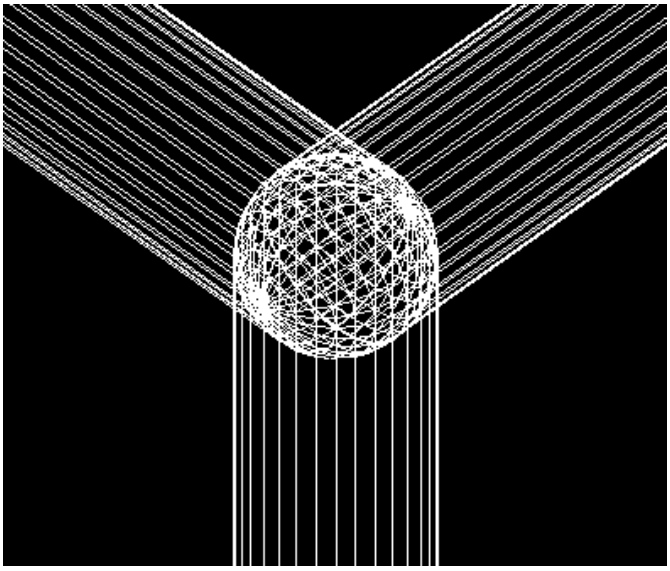
# Rounded box

Written by Paul Bourke
July 1989

Creating box shapes is very common in computer modelling applications. The boxes used to form walls, table tops, steps, etc generally have perfectly sharp edges. Such sharpness does not normally occur in real life because of wear and for safety reasons.

There are many ways of introducing curvature and ideally this would be done in the rendering phase. One modelling technique is to turn edges into cylinders and the corners into spheres. The planar facets that made up the original object are trimmed back until they are tangent to the sphere and/or cylinder surface.

To illustrate this consider the following which shows the corner of a box converted into a corner with curvature.

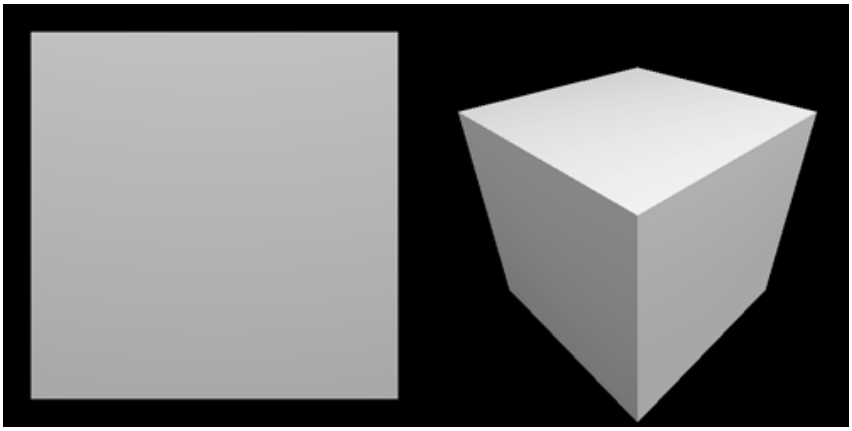


Over the whole box, each of the 6 facets reduce in size, each of the 12 edges become cylinders, and each of the 8 vertices become spheres.

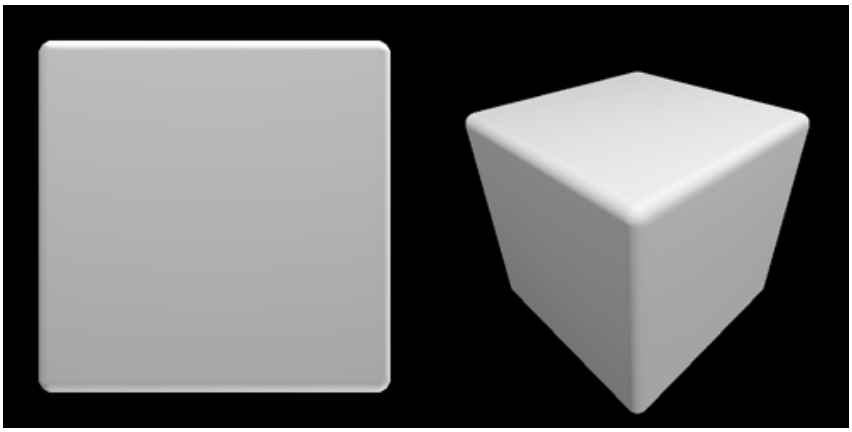
One problem with this technique as described here is that the resulting object does not normally have the desired effect internally. If this is important then the cylinders and spheres described above need to be turned into the appropriate cylindrical and spherical wedges/sections.

In the following example a cube with sides of length 2 and increasing edge radii is used to illustrate the effect.

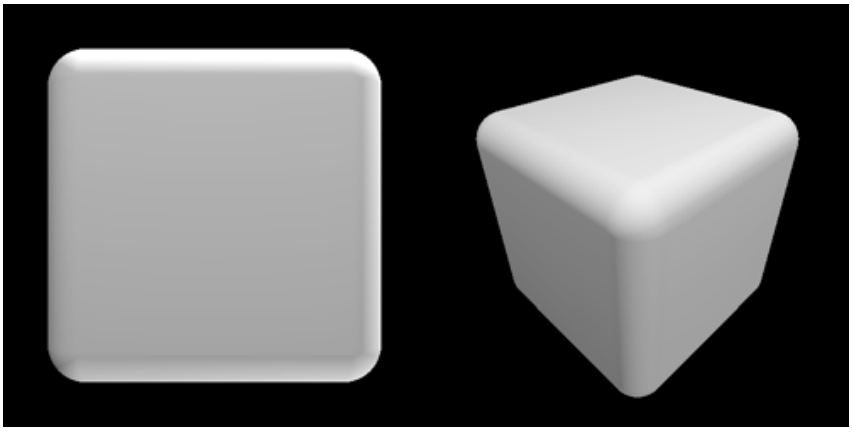
radius = 0
Just a cube



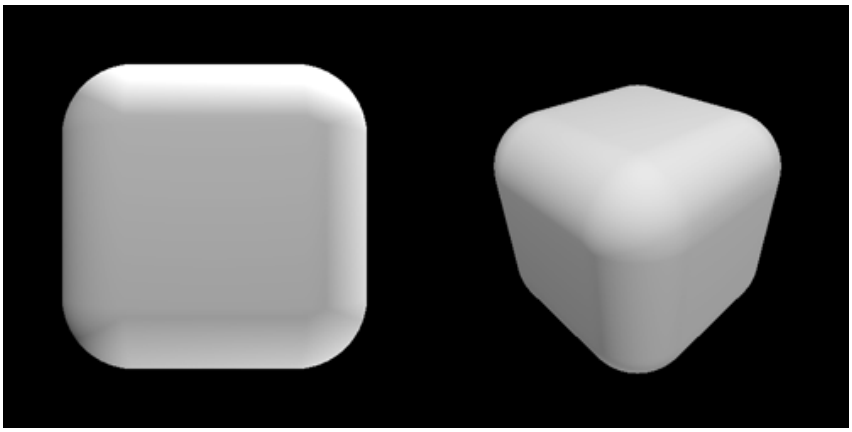
radius = 0.1



radius = 0.25

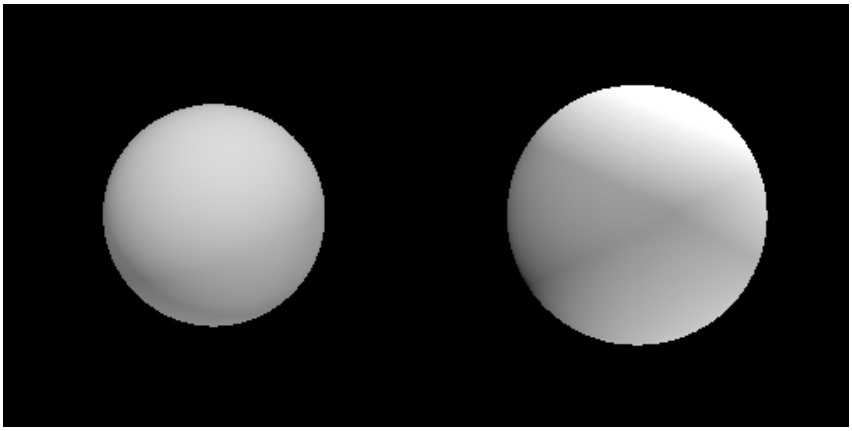


radius = 0.5



radius = 1.0

Degenerate case where the four spheres are coincident, the cylinders don't exist, and there are no facets.



# Pipes for Rendering Engines

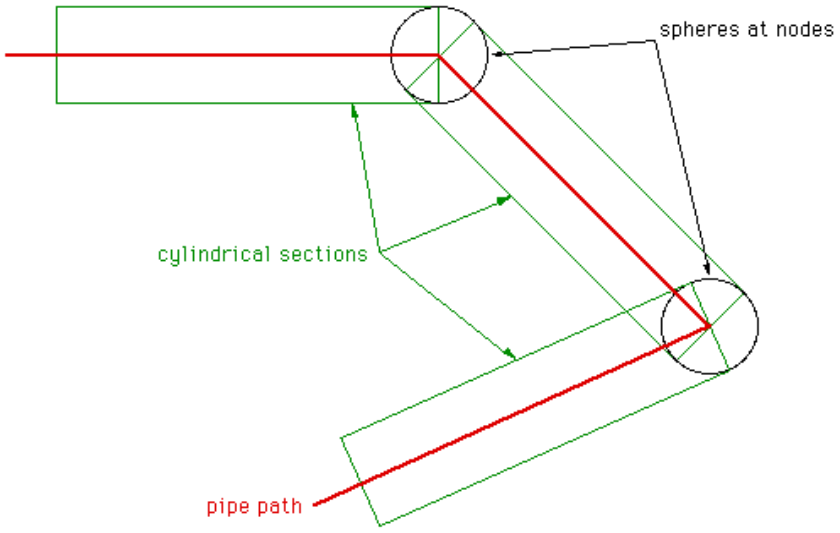
Written by Paul Bourke

June 1996

Most rendering engines support simple geometric primitives such as planes, spheres, cylinders, cones, etc. Many times a pipe is needed, by pipe I am referring to a tube like structure which passes through 3D space. The actual path is irrelevant but might be an arc or a Bezier/Spline curve defined by control points in space.

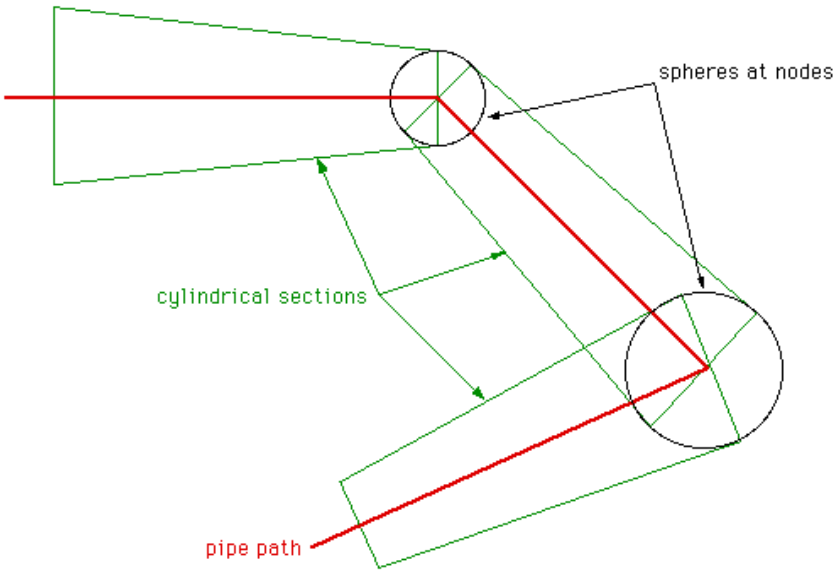
The standard method of geometrically representing this structure, as illustrated here, uses combinations of cylinders and spheres. Basically the curve is split into a straight line approximation to the desired level or resolution. Each straight line

segment is represented by a cylinder. Since this would lead to gaps at the intersection of cylinders, spheres of the same radius are placed at the intersection points. Optionally disks can be placed at the end points to seal the pipe.




**Note 1**

If the radius of the pipe is to change along the path then the cylinders need to be replaced with a cone sections, namely a cylinder with different radii at each end. The radius of each cylinder is the same at an intersection point so an appropriate sphere still fills the gaps.




**Note 2**

This method is only suitable if the pipe is to be viewed from the outside.

As an example, the following pipes are arc paths, 20 straight line sections per pipe.