

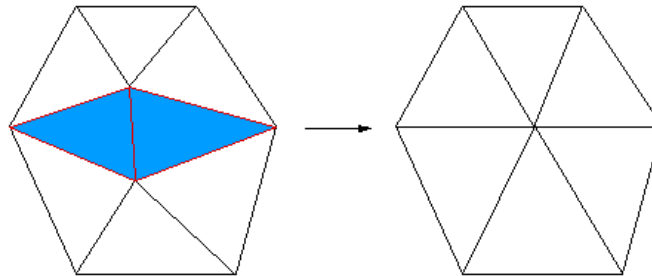
Surface (Polygonal) Simplification

Written by [Paul Bourke](#)

July 1997

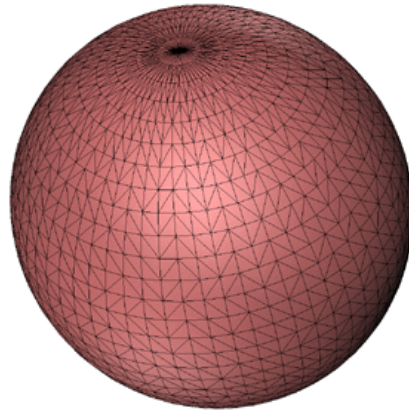
The following describes a method for reducing the number of polygons making up a surface representation while still attempting to retain the fundamental form of the surface. The applications for this are obvious if performance improvements are being sought for rendering and/or interactive environments.

The basic approach is to repeatedly remove the polygons with the shortest shared edge. Two polygons are removed on each step, the vertices of the remaining polygons are shifted to the midpoint of the shorted edge.

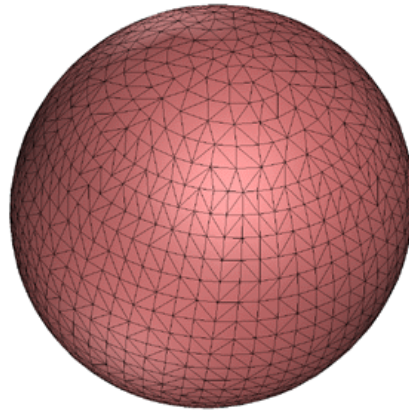


The following example illustrates the technique for a facet representation of a sphere. The initial sphere has 4000 facets, on each iteration the number of polygons is reduced by 1000. The initial sphere is obviously inefficient, to begin with, there are regions with much more detail than others (eg: the poles). If a smooth shaded rendering is being used then the model with 1000 facets is probably just as good as the original with 4 times the number of facets.

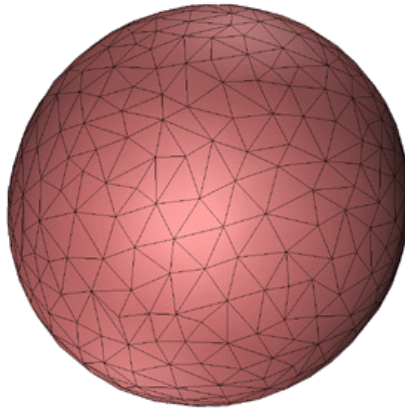
Original, 4000 facets



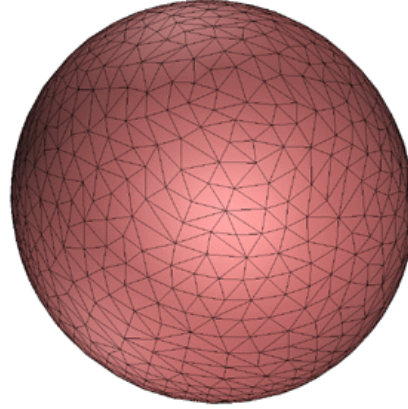
Reduced to 3000 facets



Reduced to 1000 facets

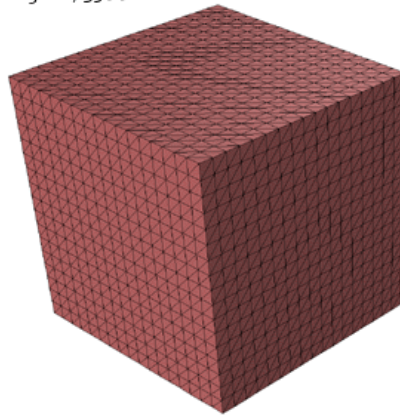


Reduced to 2000 facets

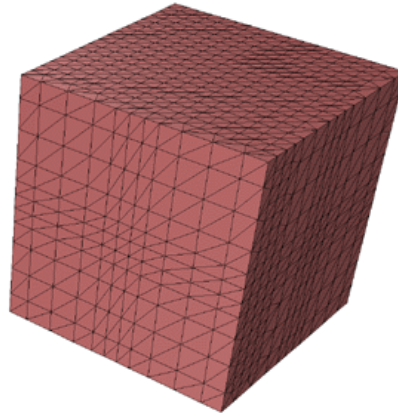


As expected, for severe reductions in the number of polygons the surface undergoes a smoothing and loss of detail. This is readily illustrated for a gridded cube.

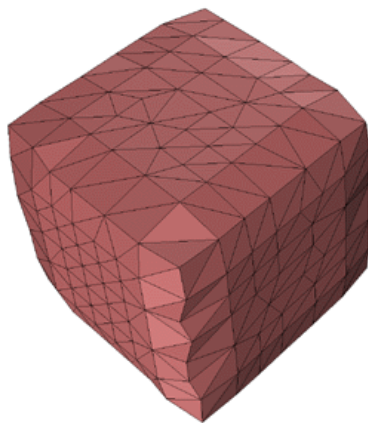
Original, 3500 facets



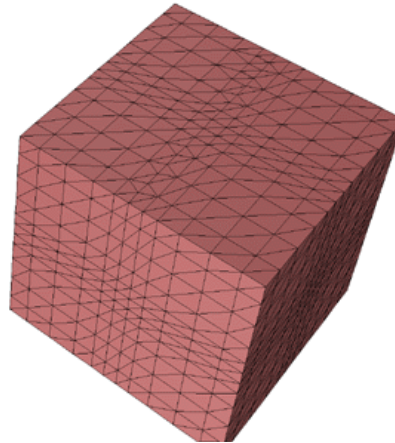
Reduced, 2500 facets



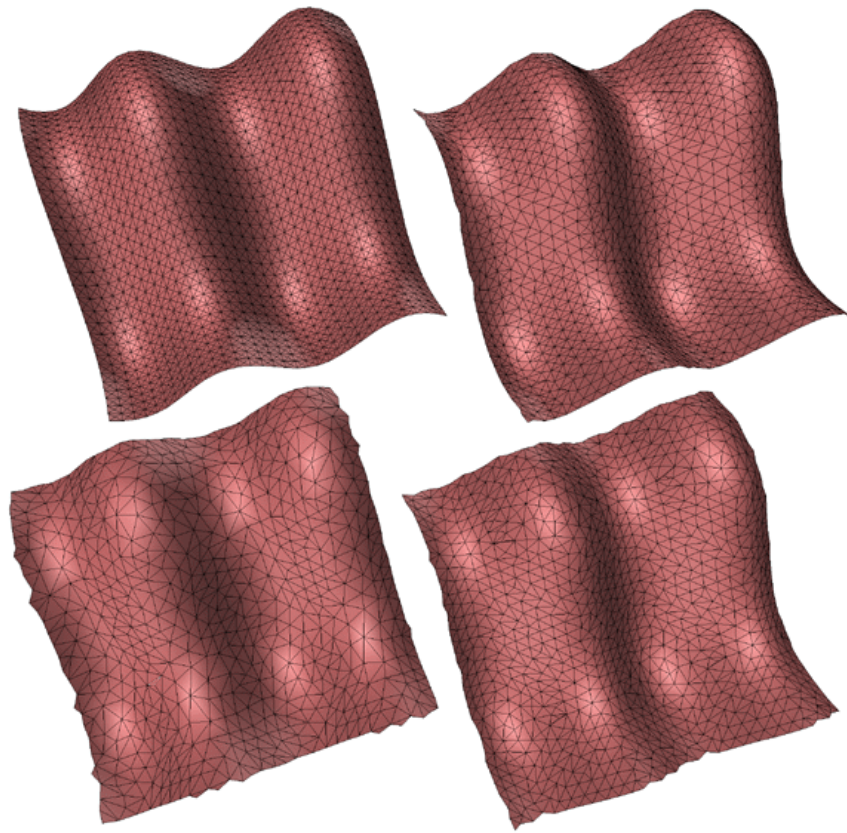
Reduced, 500 facets



Reduced, 1500 facets

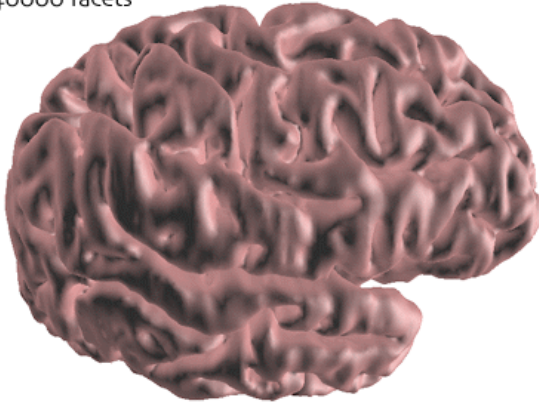


Careful consideration needs to be given to the edges of non-manifold surfaces. A straightforward implementation will slowly eat away at the edge when the shortest facet edge is on the edge of the surface.

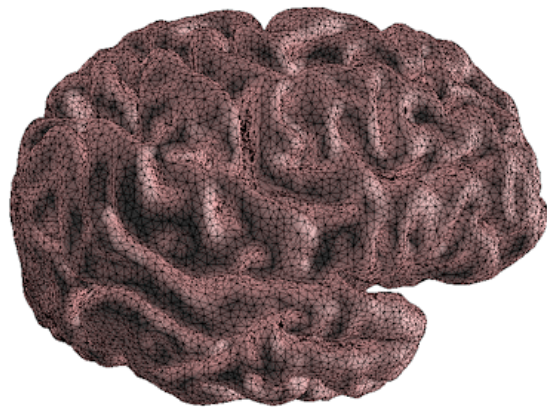
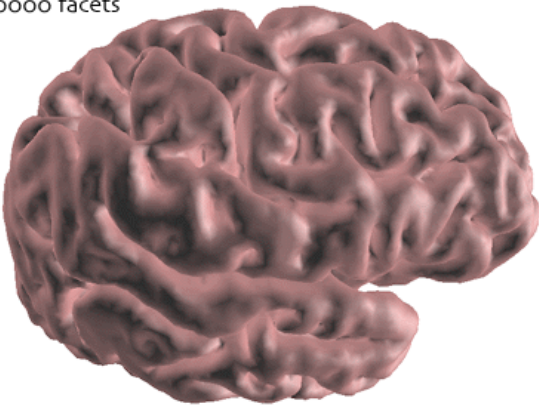


And finally, applied to a model of the human cortex.

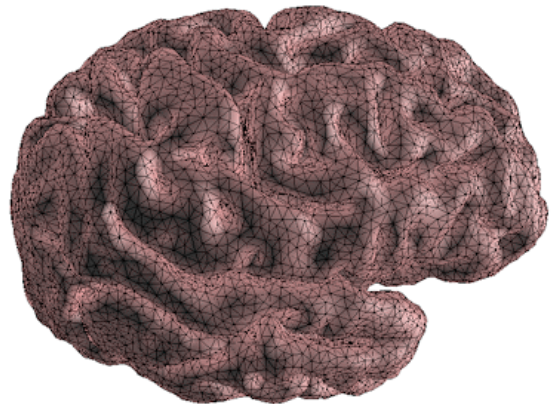
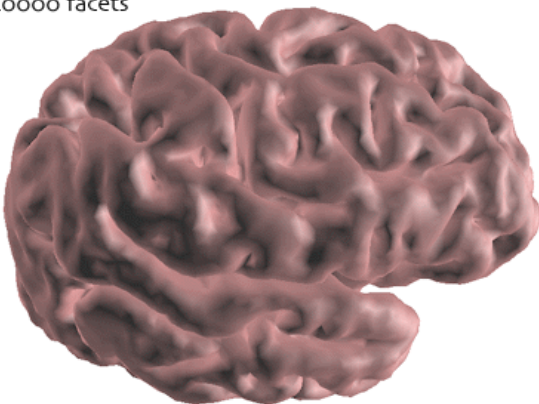
140000 facets



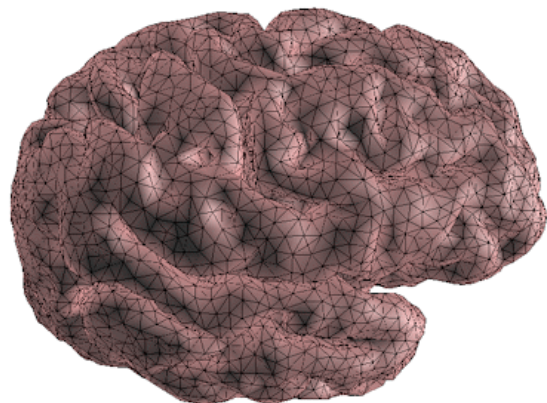
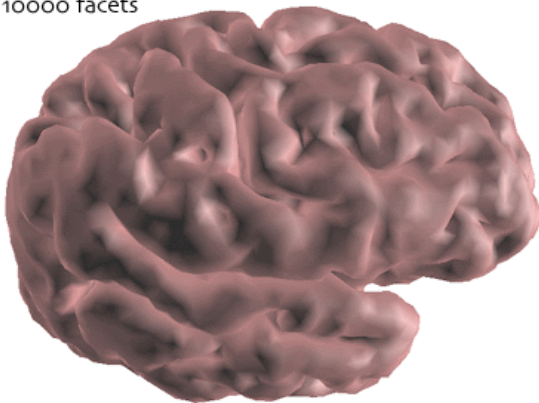
80000 facets



20000 facets



10000 facets



Clipping Polygonal Facets With An Arbitrary Plane

Written by [Paul Bourke](#)

February 1997

This note along with the source code at the end clips a 3 vertex facet by an arbitrary plane. The facet is described by its 3 vertices, the clipping plane is specified by its normal and one point on the plane. The clipping side of the plane is taken to be that one containing the normal, in the diagram below this is the right side of the clipping plane.

The standard equation of a plane is

$$A x + B y + C z + D = 0$$

where (A,B,C) is the unit normal. The value of D is determined by substituting in the known point (P_x, P_y, P_z) on the plane, namely

$$D = - (A P_x + B P_y + C P_z)$$

For a vertex (Q_x, Q_y, Q_z) the expression

$$\text{side}(Q) = A Q_x + B Q_y + C Q_z + D$$

can be used to determine which side of the plane the vertex lies on. If it is positive the point lies on the same side as the normal, if negative it lies on the other side, if zero it lies on the plane.

After determining if an edge intersects the cutting plane it is necessary to calculate the intersection point as that will become a vertex of the clipped facet. Let the edge be between two points P_0 and P_1 , the equation of the points along the line segment

$$P = P_0 + u(P_1 - P_0)$$

where u lies between 0 and 1. Substituting this into the expression for the plane

$$A (P_{0x} + u (P_{1x} - P_{0x})) + B (P_{0y} + u (P_{1y} - P_{0y})) + C (P_{0z} + u (P_{1z} - P_{0z})) + D = 0$$

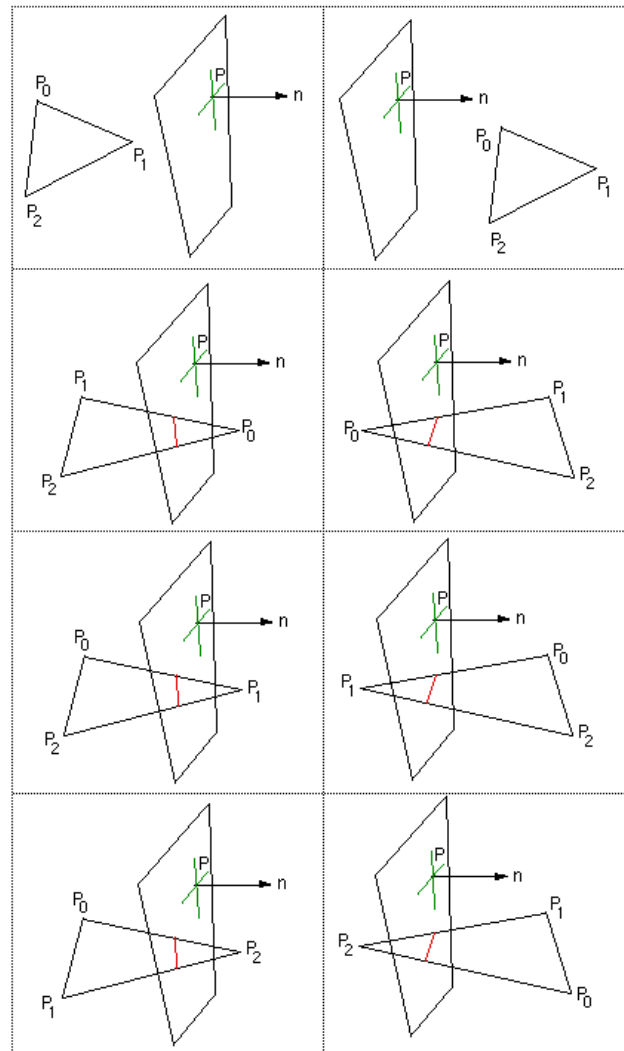
Solving for u

$$u = - \text{side}(P_0) / (\text{side}(P_1) - \text{side}(P_0))$$

Substituting this into the equation of the line gives the actual intersection point.

There are 8 different cases to consider, they are illustrated in the table below and appear in order in the source code. The top two cases are when all the points are on either side of the clipping plane. The three cases on the left are when there is only one vertex on the clipping side of the plane, the three cases on the right occur when there are two vertices on the clipping side of

the plane.

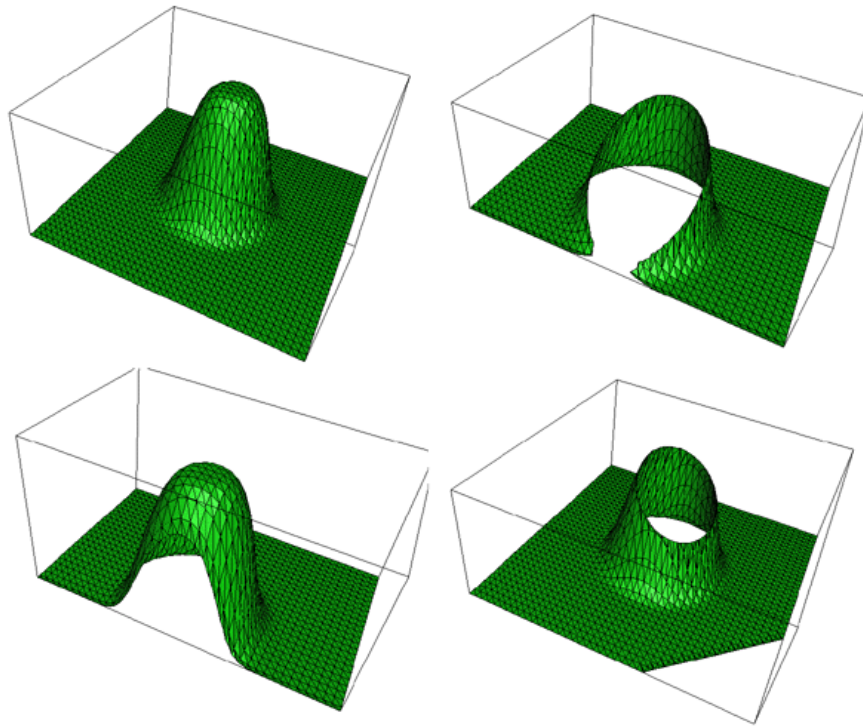


Note

- When there is only one point on the clipping side the resulting facet has 4 vertices, if the underlying database only deals with 3 vertex facets then this can be bisected using a number of methods.
- When the facets to be clipped have more than 3 vertices then they can be subdivided first and each segment clipped individually.
- The algorithm as presented has no divide by zero problems
- The source below does the clipping in place, the order in which the vertices are calculated is important for the 3 cases when there is one point on the clipping side of the plane.

[C Source](#)

As an example and testing exercise, the following shows a 2 dimensional Gaussian the the result from three clipping planes.



Surface Relaxation And Smoothing

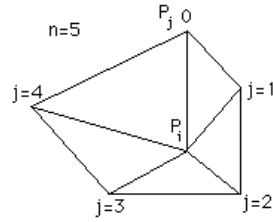
Written by [Paul Bourke](#)

January 1997

This document describes a method of smoothing a general surface described by a collection of planar facets. The facets need not be evenly spaced nor lie on a grid, they can form an arbitrary 3D surface. The method can be applied iteratively to smooth the surface to an arbitrary degree. In general the method attempts to preserve gross features and it tends to result in equal size facets (normally a desirable characteristic).

This general technique is often known as surface relaxation.

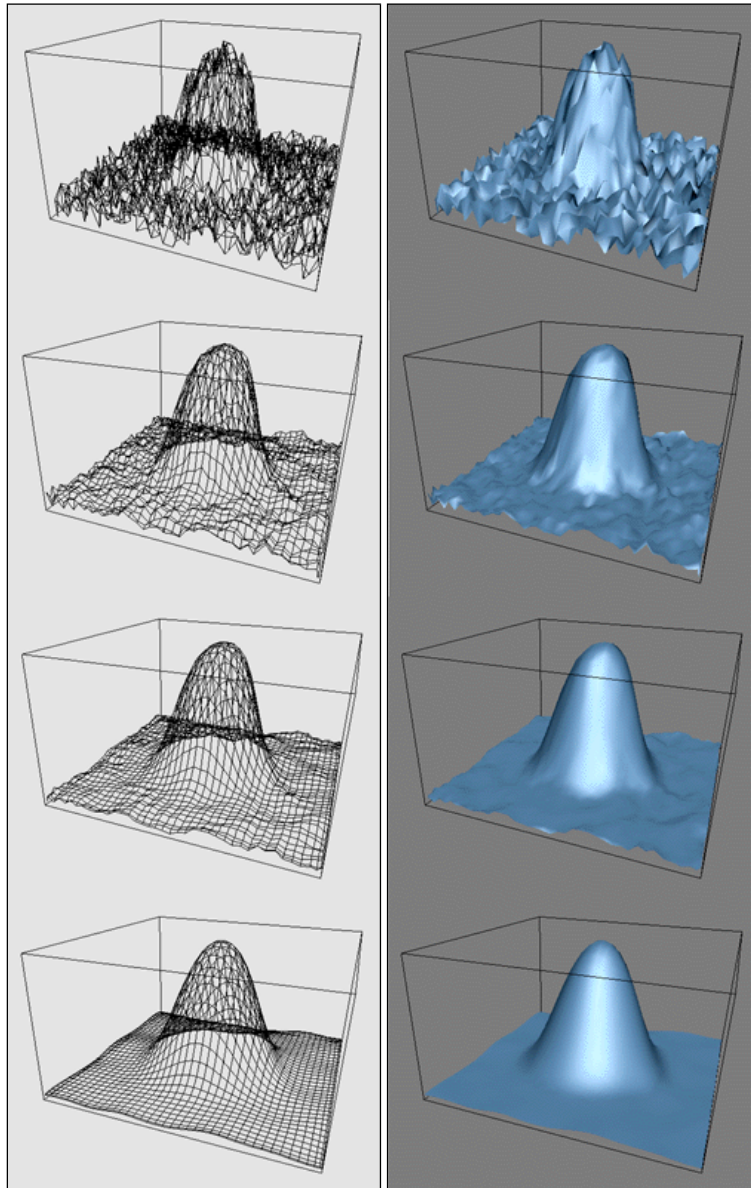
Consider a point P_i surrounded by n vertices P_j making up the facets sharing vertex P_i .



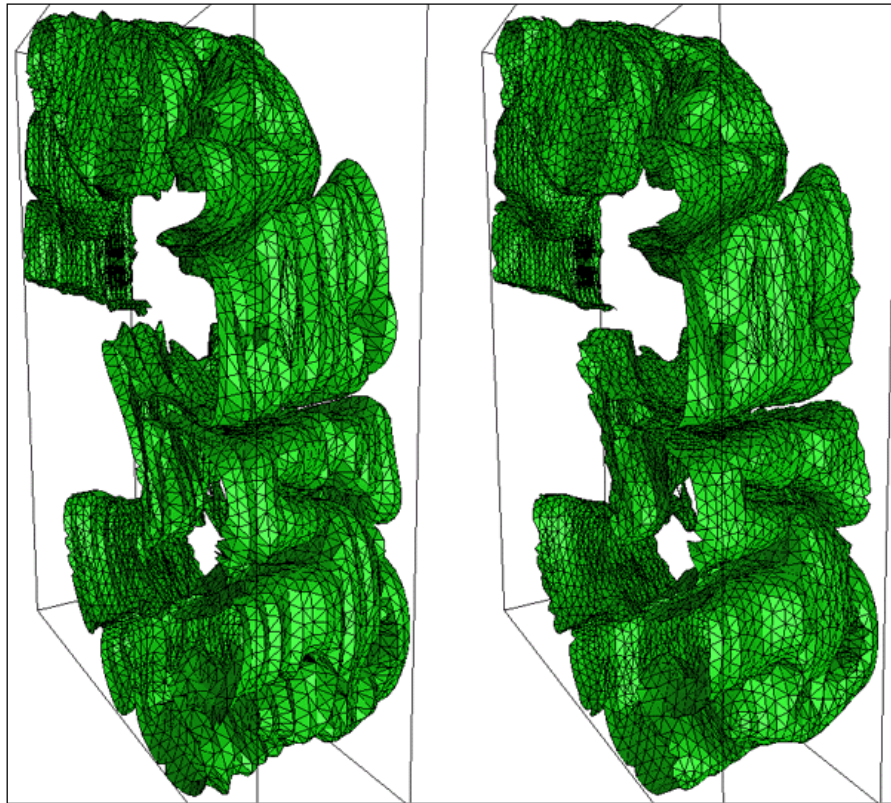
Then the point P_i is perturbed as follows:

$$P'_i = P_i + \frac{1}{n} \sum_{j=0}^{n-1} (P_j - P_i)$$

As an example consider the top figure below, it is a 2D Gaussian with noise added to each x,y,z component of each vertex. The subsequent smoothed versions are the result of iteratively applying the technique 1,2,3 and 6 times respectively.

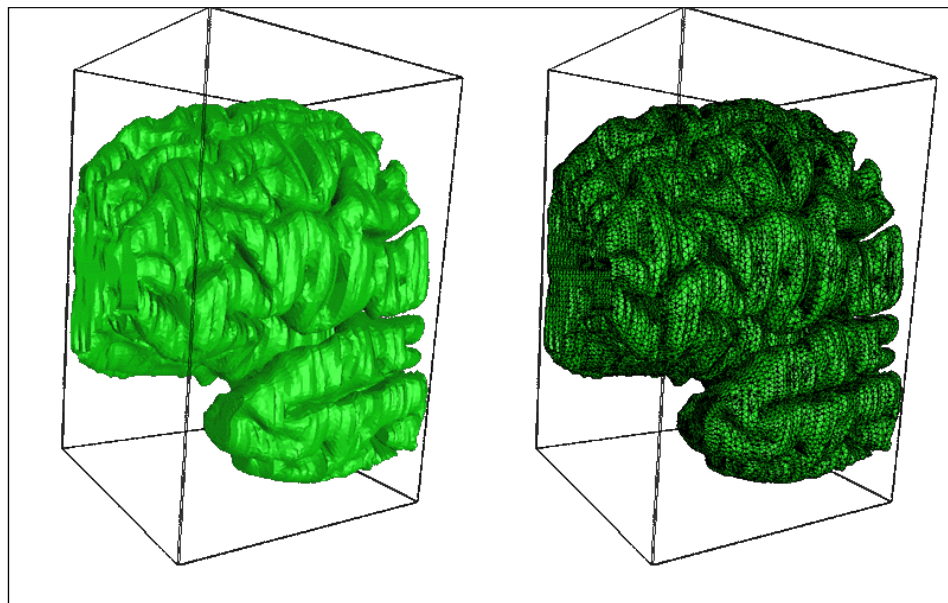


The following is a more "real life" example, in particular, the facets are not arranged on a simple grid and are of varying sizes. The geometry is a 2cm slice from a model of the human cortex. The original surface on the left has clear vertical ribs which arose as a result of digitising scanned images which were slightly misaligned. The surface on the right has one iteration of relaxation applied to it. The vertical artifacts are clearly reduced.

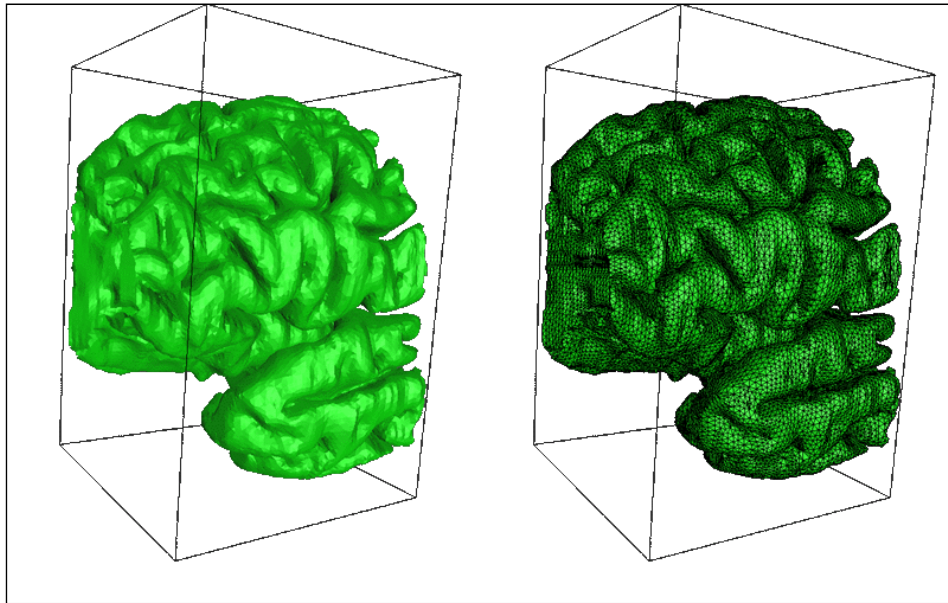


The following shows a different sectioning of the same model through two relaxation iterations. The pairs of images are the same except that the facet edges are drawn on the surface on the right. In both cases the surface has been rendered using flat shading, that is, without smoothing across facets.

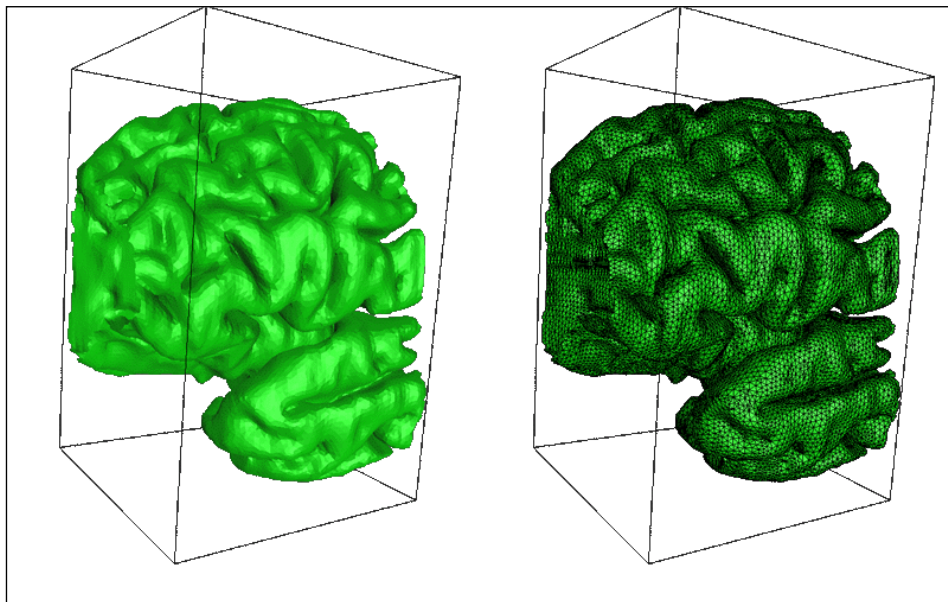
Original surface



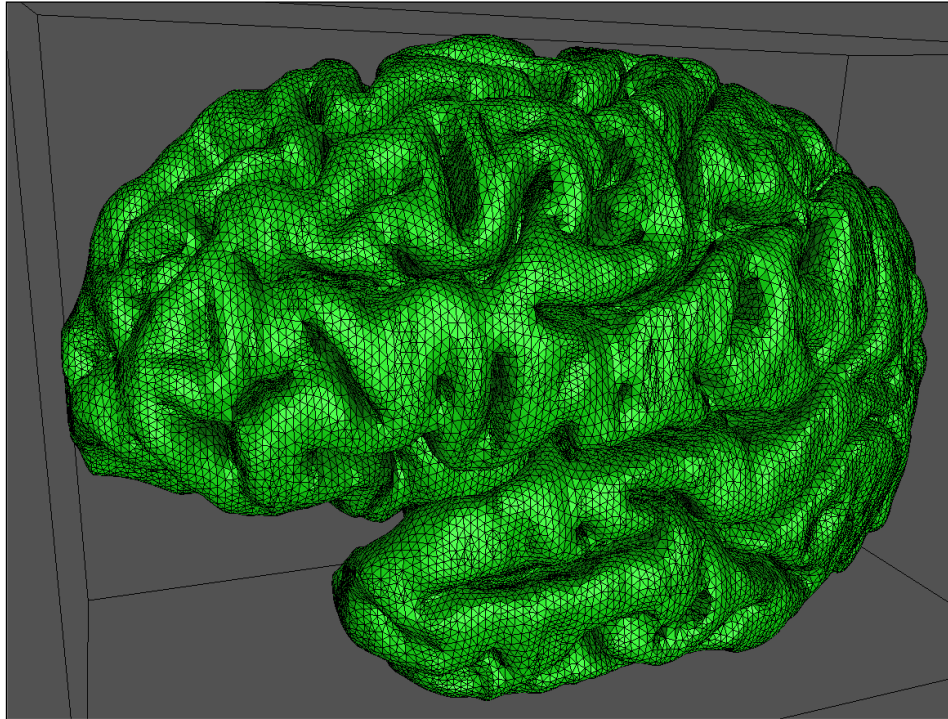
One degree of relaxation



Two degrees of relaxation



And finally the whole surface



Geometric Crumpling

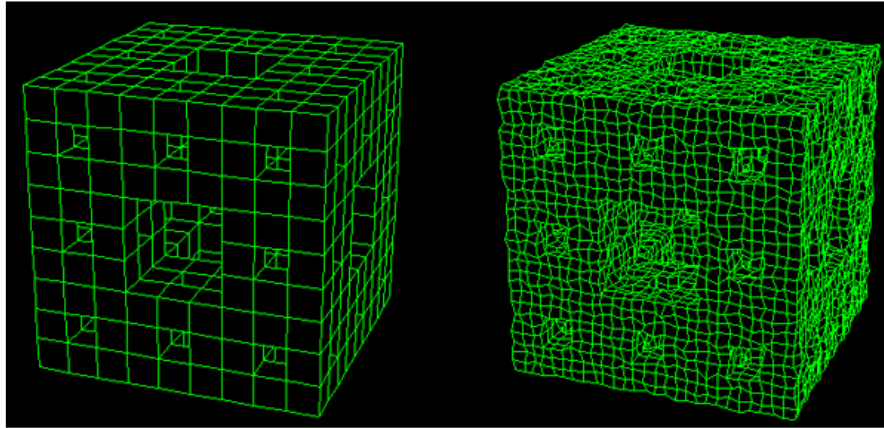
Written by [Paul Bourke](#)

January 1994

In most rendering applications rough surfaces can be simulated with what are known as texture maps or more precisely bump maps. These are images which perturb the surface normal during the rendering process creating the appearance of a rough surface or raised and lowered structure such as grouting in a tiled floor.

In some instances it is desirable to actually represent the surface roughness geometrically. For example, many textures don't retain their desired effect when viewed from very close. Some effects are also hard to generate using bump maps, for example, the extent of the deformation is limited.

There are many ways of modifying the geometry depending on the exact effect. One effect is that of crumpling, what you might do to a piece of paper. In the following example, each facet is split into a 3x3 grid. Each vertex is then displaced by a random amount (Gaussian distribution). Of course for each displacement all facets which share that vertex are also modified.



The above shows a menger sponge before and after the crumpling exercise.

There are a few things to note

- * This whole process is inefficient to say to least. In this case there has almost been a 10 fold increase in the geometric content.
- * 4 point facets are in general no longer coplanar even if they were in the original model, triangulating these further increases the geometric content.
- * The extent of the displacement is limited if it is undesirable for facets to intersect.

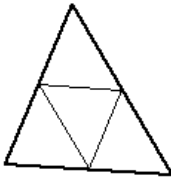
Splitting Facets

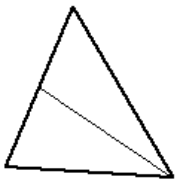
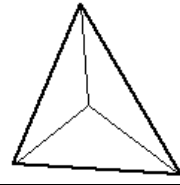
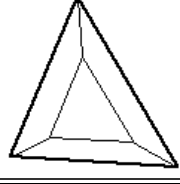
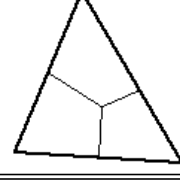
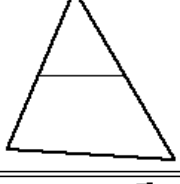
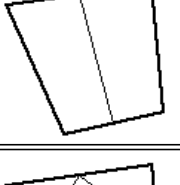
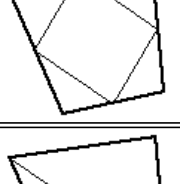
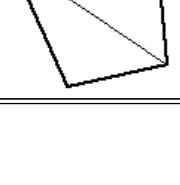
Written by [Paul Bourke](#)

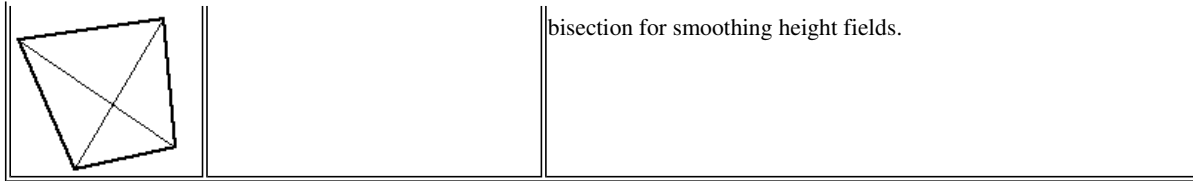
November 1991

The following discusses a number of ways of splitting up facet based computer models into smaller facets. There are a number of reasons for wanting to do this, some include:

- In order to add detail to the surface, for example, fractal spatial subdivision
- For surface relaxation algorithms
- To eliminate narrow angled facets which cause problems for some rendering and texture algorithms

Method	Additional Goemetry	Comments
	Increases the number of facets by 3	Perhaps the most common splitting technique. Doesn't result in finer internal angles.

	Doubles the number of facets	Simplest method, bisection. Normally the longest edge or the widest internal angle would be bisected. While it reduces long edges it also tends to produce narrow internal angles
	2 additional facets	The centroid becomes the new vertex. If the facet is part of a height field the centroid height would normally be the average of the 3 original vertex heights.
	Results in an identical but smaller triangular facet and three new 4 vertex facets.	Uncommon
	Yields three 4 vertex facets.	The centroid is usually used as the mid point.
	Gives a smaller 3 point facet and a new 4 point facet.	This is a common first approach for long thin facets, the cut is made along the two longest edges
	Subdivide a 4 point facet into two 4 point facets	Simple example of a more general repeated bisection of a facet. The longest opposite pair of edges are split if equal size facets are desirable.
	Results in a small 4 point facet and four new triangular facets.	Uncommon
	Bisection into two triangular facets	The simplest splitting of a 4 point facet. Most commonly used by rendering program to ensure all facets are planar. Results in facets with small internal angles.
	Gives three triangular facets.	The centroid is normally used. An improved technique to simple

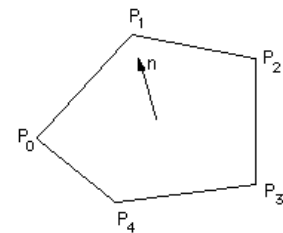


Facets, Planes, Normals, Rendering

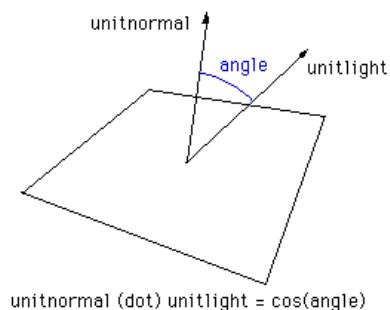
Written by [Paul Bourke](#)

November 1992

The usual way of representing a bounded planar surface (facet) in computer graphic applications is as a sequence of vertices. Shading algorithms and raytracing generally requires knowledge of the normal to the facet, this is calculated by taking the cross product two of the edge vectors of the facet. The angle the normal makes with the light source vector determines the degree of shading of the facet. In particular, if the normal points towards the light source then the surface is brightly illuminated, if it points away from the light source then the surface is in shadow.



A common problem arises because the vertices need then to be specified in a specific order. A common convention is they need to be ordered such that the normal points outwards. This assumes there is an "outer" and "inner" ie: that the object is closed.

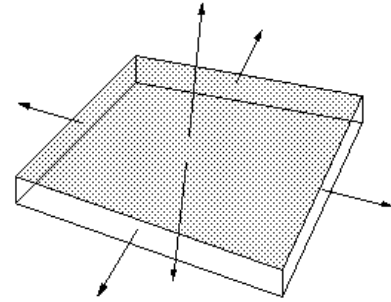


The usual way of calculating the angle between the normal and light source vector involves taking the cross product between these two vectors giving the cosine of the angle between them. Some rendering packages simply use the absolute value of this angle thus shading the back side the same as the side facing the light source.

Another technique which is common when you don't have control over the source of the facets nor the rendering package is to double up each facet, the duplicate having its vertices in the reverse order. These two sided facets can obviously make up objects which are not closed.

The basic problem arises because facets/planes don't exist in real life, all planar surfaces have a finite thickness. This is similar to the issue of representing lines in raytracing packages....they must be turned into objects with finite thickness

such as cylinders. Creating planes with a finite thickness has been recognised for a long time in Architectural modelling where creating walls from infinitely thin planes leads to all sorts of problems not encountered if their true thickness is used.



Polygon Types

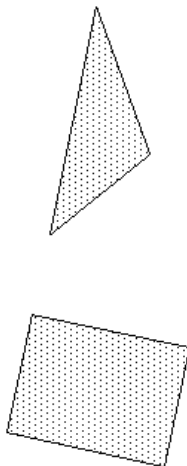
Written by [Paul Bourke](#)

January 1993

There are a number of categories of polygons in common usage in computer modelling and graphics. The particular polygon type being used can have a dramatic effect on the complexity of many rendering and editing algorithms. For example, an algorithm for splitting a polygon into a number of 3 vertex facets is trivial for convex polygons and quite problematic for polygons with holes.

While most of the discussion will be with regard to polygons as bounded planes in 3D, the same ideas apply to polygons in 2D.

Some of the more frequently used categories will be listed and discussed below.



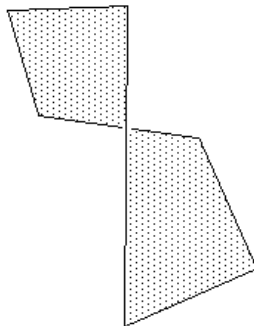
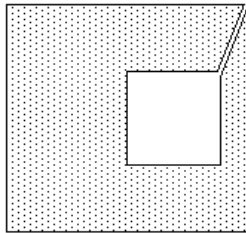
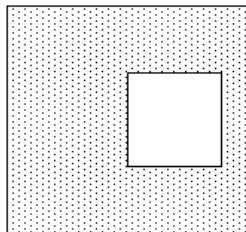
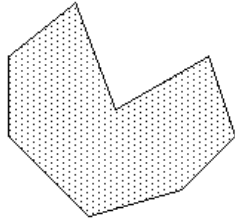
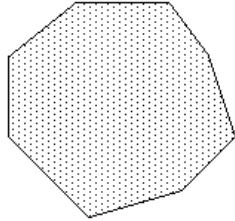
3 vertex facet

This is the simplest type of polygon. Perhaps one of the most important characteristics is the points lie on a plane, as such it is often the most fundamental primitive for 3D rendering applications which expect a single unambiguous normal for the whole polygonal region.

Rectangular facet

Rectangular or 4 vertex polygons are generated from gridded datasets and polygonal approximations of 2D surfaces. Many applications where such rectangular facets arise naturally don't ensure the vertices are coplanar, fortunately it is trivial to turn such a polygon into triangular and hence planar polygons.

Convex polygon



This is the simplest type of polygon with more than 3 or 4 vertices.

Concave polygon

This is the most general type of "simple" polygon, that is, without holes.

Complex polygon

Polygons with holes are defined in a number of ways. The holes can be "tagged" as such, a common method is to define the vertices of the holes in a order different from that of the solid parts. For example the solid parts might be defined clockwise about the normal, the holes anticlockwise about the normal.

Such polygons can be turned into concave polygons by introducing 2 coincident edges, between the solid and hole polygons. For multiple holes and concave solid pieces the coincident edges need to be taken between appropriate vertices to avoid making an overlapping polygon.

Intersecting polygon

These somewhat perverse polygons normally get lumped in with complex polygons with holes. The example on the right is the most straight forward case, situations where the polygon covers part of itself are generally avoided and are not handled consistently by rendering engines.