Point Based Rendering of Non-Manifold Surfaces with Contours

Ron J. Balsys*
Faculty of Informatics and Communications, Central Queensland University, Rockhampton M.C., Qld. 4702, Australia
Kevin G. Suffern†
Faculty of Information Technology, University of Technology Sydney, P.O. Box 123, Broadway NSW 2007, Australia

Abstract

We present point based rendering techniques that render various types of contours as constant width slabs on surfaces. The techniques requires evaluations of the surface functions and gradients to render shaded images. We use slabs parallel to the principle planes, slabs located along a principal axis and rotated by arbitrary steps, slabs consisting of concentric spheres and slabs of constant Gaussian and mean curvatures. We also use the technique to render curvature maps of surfaces. We illustrate the techniques with a number of parametric and implicit surfaces, and discuss their advantages and disadvantages compared to other rendering techniques.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

Keywords: point rendering, countours, curvature, radial planes-point rendering, contours, Gaussian and mean curvature, curvature maps, parametric surfaces, implicit surfaces

1 Introduction

There are two common methods for representing surfaces mathematically in $\Re^3$. One is parametrically with equations of the form

$$x = x(u,v),\ y = y(u,v),\ z = z(u,v), \tag{1}$$

where u and v are the parameters and x, y and z is a point in $\Re^3$. The other is implicitly with a single equation of the form

$$f(x,y,z) = 0. \tag{2}$$

Surfaces can be regular, singular, or non-manifold. Singular surfaces include those that self intersect, and non-manifold surfaces have regions where they are locally non Euclidean, for example, points of infinite curvature.

There are a number of techniques in common use for rendering surfaces, the main ones being polygonization, scan line, ray tracing, and point based techniques. They all have their advantages, disadvantages, and limitations. Regular parametric surfaces are usually straightforward to polygonize to arbitrary precision, enabling them to be rendered by standard polygon rendering techniques, see Foley et al. [1990]. However, the presence of singularities and non-manifold features can create difficulties. For example, regions of arbitrarily large curvature can require large numbers of polygons for accurate representation.

Regular implicit surfaces can also be polygonized for rendering, but the process is not as straightforward as it is for parametric surfaces. A large number of papers have been published on the polygonization of implicit surfaces. Relevant examples include: Wyvill et al. [1986], Lorenson and Cline [1987], Bloomenthal [1988], Schmidt [1990] and Balsys and Suffern [2001]. Thin sections, even if they are regular parts of surfaces, can also be difficult to polygonize.

Scan line techniques can render parametric and implicit surfaces with singularities and non-manifold features, but can be complex to program, see for example Foley et al. [1990] and Sederberg and Zundel [1989]. Ray tracing can also render parametric and implicit surfaces with singularities and non-manifold features, but the ray-surface intersection equation has to be solved for each surface rendered. Algebraic implicit surfaces can be ray traced if a robust polygon solver is available, see Hanrahan [1983], but for non-algebraic surfaces it can be difficult to solve the ray-surface intersection equation.

Point based rendering techniques use points, or objects such as discs, as rendering primitives. Levoy and Whitted [1985], and Rockwood [1987] are early papers on rendering parametric surfaces using points. Witkin and Heckbert [1994] used repulsive particles called floaters that can slide over implicit surfaces for their modeling and rendering. The repulsion creates uniform distributions of particles on the surface and these are rendered as discs.

De Figueiredo and Gomes [1996] used physically based particles that obey Newtonian equations of motion to render differentiable implicit surfaces. They used points for rendering, but did not produce shaded images. Rosch at al. [1997] used large discs to interactively render self-intersecting, singular, algebraic implicit surfaces with non-manifold points (cusps).

Recent work by Tanaka et al. [2000], [2001] used particles that obey stochastic differential equations to render twice differentiable implicit surfaces. Other than the differential constraint, the surfaces are arbitrary, the points are evenly distributed over the surfaces, and surface intersections can also be rendered. These are nice features of their work, but the method is complex, and would be a lot of work to implement from scratch. Shaded images of single surfaces are produced with discs.

Jones et al [2003] presented a point based rendering technique for parametric and implicit surfaces that is based on iterated function systems [Jones and Moar 2000], [Jones 2001]. It's main advantage over previous point based rendering techniques is it's simplicity. It can also render singular and non manifold surfaces, and non algebraic implicit surfaces.

Here, we extend the technique of Jones et al. [2003] to render surfaces with contour lines. These can be used to help visualize the surfaces themselves, or render functions defined on them. The contour lines are the intersections of the surfaces with the level surfaces of scalar fields $g(x,y,z) = c$. As examples, we use the scalar

*e-mail: balsys@cqu.edu.au
†e-mail:kevin@it.uts.edu.au

fields $g_1 = x$, $g_2 = y$, $g_3 = z$ whose level surfaces are planes parallel to the coordinate planes, or use a series of planes rotated about a coordinate axis. The functions defined on the surfaces include the Gaussian and mean curvatures. We also produce curvature maps of surfaces.

The contouring part of the algorithm is adapted from Balsys and Suffern [2003] where we rendered contours on ray traced surfaces. The algorithm we modify is the slab algorithm which renders the contours as bands of constant finite width on the surface.

The algorithms presented here can be classified as surface interrogation methods, for which there is a large literature. The following are surveys: Hoitsma and Roche [1983], and Lee and Fredericks [1984], Satterfield and Rogers [1985], Hagen, Schreiber and Gschwind [1990], Higashi, Kushimoto and Hosaka [1990], Hagen, Hahmann, Schreiber, Nakajima, Wördenweber and Hollemann [1992], and Maas [1994]. Methods for producing surface contours for general parametric surfaces are given in Petersen [1984].

2 Point Based Surface Rendering

We use an algorithm that has points as the display primitive. Much of what follows in this section has been reported before but we repeat it now for completeness. For parametric surfaces we generate points in the 2D parameter space (u, v) which gives us 3D points, (x, y, z), on the surface. The point's z value is checked against the current z buffer value and if the point has a smaller z value, in viewing coordinates, it is shaded using Phong shading modified to draw contours. A shadow buffer can also be incorporated so that shadows are rendered as given in Foley [1990].

The method uses random numbers to stochastically generate points in the plotting volume. We use the iterated function system (IFS) approach from Jones and Moar [2000] to generate stochastically distributed points in the (u, v) parameter space. Algorithm 1 shows the details. If the number of iterations of the IFS system is large enough the surface will be fully rendered. Hidden point, point shadow and shading are then applied.

Algorithm 1. Iterated function system generation of points for parametric surfaces.

```
double a = 0.5; b = 0.5, u, v; // a, b, in [0..1]
double minU, maxU, minV, maxV;
long loops;
Point3D f;
// pF returns point on surface F
Function* pF;

for (int k = 0; k < loops; k++) {
   a = (((arand32 & 2) >> 1) + a) / 2;
   b = ( (arand32 & 1) + b) / 2;

   u = minU*(1.0-a) + maxU*a;
   v = minV*(1.0-b) + maxV*b;
   f = pF(u, v);
   Plot(f);
 // 32 bit registers & 2 bits for selection
   if (k % 16 == 0)
      arand32 = rand();
   else
      arand32 = arand32 >> 2;

}
```

Figure 1(a) shows the parametric triaxial tritorus surface [Bourke, 2003],

$$\begin{aligned} x &= \sin(u)\,(1+\cos(v)), \\ y &= \sin\left(u+\frac{2\pi}{3}\right)\left(1+\cos\left(v+\frac{2\pi}{3}\right)\right), \\ z &= \sin\left(u+\frac{4\pi}{3}\right)\left(1+\cos\left(v+\frac{4\pi}{3}\right)\right) \end{aligned} \tag{3}$$

where $0 \le u \le 2\pi$ and $0 \le v \le 2\pi$, rendered using points.

For parametric surfaces we needed to define two parameters, (u, v), to generate a point (x, y, z) on f, but for implicit surfaces we must find the three values of (x, y, z) for which

$$f(x, y, z) \le |\varepsilon|. \tag{4}$$

The IFS method picks points in a cube rather than in a square. The value of ε to use will depend on the surface. Essentially, infinitesimal points are replaced by "spherical points" whose diameters are ε in parameter space. If ε is small the "spherical point" will be represented by at most 1 or 2 pixels on screen. If this "spherical point" is too large then the surface will be poorly rendered. As the surface function f, the clipping volume and the number of pixels on the display is known, an estimate for a starting value for ε can be readily calculated. Increasing ε reduces the number of points required to cover the surface at the expense of accuracy. Essentially the rendered surface has thickness ε using this method.

We use the 3D IFS algorithm described in Jones et al. [2003]. Algorithm 2 details the point generation process in this case.

Algorithm 2. Iterated function system generation of points for implicit surface.

```
Point3D p;
Double a, b, c;
Point3D p;
// pF returns value of F(p) on surface F
Function* pF;

 int arand2 = rand();
 a = 0.5; b = 0.5; c = 0.5;

 for (int k = 0; k < loops; k++) {
    a = (((arand2 & 4) >> 2) + a)/2;
    b = (((arand2 & 2) >> 1) + b)/2;
    c = ((arand2 & 1) + c)/2;
    p.x = leftX*(1.0-a) +rightX*a;
    p.y = bottomY*(1.0-b) + topY*b;
    p.z = BackZ*(1.0-c) + frontZ*c;
 f = pF(p.x, p.y, p.z);
 if ( abs(f) < epsilon) { // point on surface
    Plot(p);
    }
// 3 bits used for IFS choice
 if (loops % 10 != 0)
     arand2 = arand2 >> 3;
 else
     arand2 = rand();
 }
```

Figure 1(b) shows the implicit super-toroid surface [Barr, 1981]

$$f(x,y,z) = \left(\left(\frac{x}{a}\right)^{\frac{2}{\varepsilon_2}} + \left(\frac{y}{b}\right)^{\frac{2}{\varepsilon_2}}\right)^{\frac{\varepsilon_2}{\varepsilon_1}} + \left(\frac{z}{c}\right)^{\frac{2}{\varepsilon_1}} - \left(\frac{d}{\sqrt{a^2+b^2}}\right)^{2} \tag{5}$$

with $a = b = c = 5, d = 20, \varepsilon_1 = 2$ and $\varepsilon_2 = 1$, rendered using this algorithm.

3 Rendering Slab Contours on Surfaces

In Balsys and Suffern [2003] we developed a slab algorithm that renders contours as slabs of constant specified width w on ray traced surfaces. This algorithm is independent of surface representation. Here, we discuss how we modify the slab algorithm to render slabs using points. The slab algorithm treats each of the level surfaces $g(x, y, z) = c$ as a slab of finite thickness d. We color the surface with the slab color if the point is inside the slab, otherwise the surface is colored using Phong shading.

As discussed in Balsys and Suffern [2003] we use slabs whose thickness d varies in such a way that the width w is constant. If w is the specified width, the slab thickness d is

$$d = \frac{w}{\sqrt{1 - (n_f \cdot n_g)^2}} \tag{6}$$

where n_f is the unit normal to surface, and $n_g = \frac{\nabla g}{|\nabla g|}$ is the unit normal to the slab, both evaluated at the surface point. We assume the slab is approximately planar in the neighborhood of the point. In regions of high curvature of g this assumption can break down.

For slabs parallel to the coordinate planes it's simple to determine if a point is in one of the slabs since the slabs are not curved. We only need to check if the x, y, or z coordinate of the point is within $\pm d/2$ of one of the specified x, y, or z center's of the slabs. For slabs that are curved, such as the surfaces of constant Gaussian curvature, $g(x, y, z) = K(x, y, z)$ where $K(x, y, z)$ is the Gaussian curvature of the surface being rendered, it's more complex to determine if a point is in one of the slabs. As discussed in Balsys and Suffern [2003] the distance s between the point on the surface, p, and the closest point at the centre of the contour point q, is approximately

$$s = \frac{|g(p) - c|}{|\nabla g(p) \sin\theta|} \tag{7}$$

provided p is close to q. (The distance s is only approximate because it's measured in the tangent plane to f at p, and not over f.) Close in this context means that s is small compared to the radius of curvature of the contour at q and the minimum radius of curvature of f at q.

The value of s is calculated for each contour to be plotted, and the surface is colored with the contour color if $s < w/2$ for any contour value. This process will produce contours of approximately constant width provided w is small compared to the radius of curvature of the contour. Sometimes this approximation breaks down as the radius of curvature can be arbitrarily small.

4 Surface Examples

We present a number of examples to illustrate the utility of rendering contours on surfaces.

The Charges surface

$$f(x, y, z) = \sum_{i=1}^{i=n} \frac{q_i}{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} - c \tag{8}$$

is an example of an implicit surface that is non-algebraic and therefore difficult to ray-trace. Figure 2(a) shows contours parallel to the principle planes on the surface of the Buckyball (Carbon^{60}). Figure 2(b) shows slabs that are sequences of spheres of increasing radius r_i which aids the visualization of the shape of the Buckyball surface.

Figure 3 shows the shell or ram's horn surface [Nordstrand, 2001] given by the parametric equations

$$\begin{aligned} x &= a\left(1 - \frac{v}{2\pi}\right)\cos(nv)(1 + \cos(u)) + c\cos(nv) \\ y &= a\left(1 - \frac{v}{2\pi}\right)\sin(nv)(1 + \cos(u)) + c\sin(nv) \\ z &= b\frac{v}{2\pi} + a\left(1 - \frac{v}{2\pi}\right)\sin(u) \end{aligned} \tag{9}$$

with a = 0.2, b = 1, c = 0.1, n = 2, $0 \le u \le 2\pi$ and $0 \le v \le 2\pi$. In Figure 3(a) we render contours parallel to all the principle planes and in Figure 3(b) we selectively draw the contours parallel to the z principle plane.

In Figure 4(a) we render the cyclide surface [Chandru et al. 1989]

$$\begin{aligned} f(x, y, z) &= (x^2 + y^2 + z^2)^2 \\ &- 2(x^2 + r^2)(f^2 + a^2) - 2(y^2 - z^2)(a^2 - f^2) \\ &+ 8a f r x + (a^2 - f^2)^2 \end{aligned} \tag{10}$$

with $a = 10$, $r = 2$ and $f = 2$. Here we render the surface slabs passing through the z axis and successively rotated by 15° about the z axis, as these are optimally oriented for visualizing this particular surface.

In Figure 4(b) we show contours on the surface of the Klein bottle defined in two parts [Bourke, 2001]. With $(0 \le v \le \pi)$, and $r = 4(1 - \frac{\cos(u)}{2})$, the first part is for $(0 \le u \le \pi)$,

$$\begin{aligned} x &= 6\cos(u)(1 + \sin(u)) + r\cos(u)\cos(v) \\ y &= 16\sin(u) + r\sin(u)\cos(v) \\ z &= r\sin(v) \end{aligned}$$

For the remaining range of u, $(\pi \le u < 2\pi)$,

$$\begin{aligned} x &= 6\cos(u)(1 + \sin(u)) + r\cos(v + \pi) \\ y &= 16\sin(u) \\ z &= r\sin(v) \end{aligned} \tag{11}$$

In Figure 4(b) the Klein Bottle is clipped and drawn with slabs passing through the z axis successively rotated by 15° about the z axis to reveal its inner structure.

Gaussian and Mean curvatures of surface are of interest in surface analysis. In Figure 5(a) we show the Gaussian curvature $(g(x, y, z) = K(x, y, z))$ of the ellipsoid surface [Spivac, 2003] where

$$K(x, y, z) = \frac{1}{a^2 b^2 c^2}\left(\frac{x^2}{a^4} + \frac{y^2}{b^4} + \frac{z^2}{c^4}\right)^{-2} \tag{12}$$

where a, b, and c are the semi-axes of the ellipsoid.

To choose the contour values we need to estimate the range of the scalar field over the rendered part of the surface. For implicit surfaces we do this by first rendering the image and evaluating the scalar field at each point, and for parametric surfaces we sample the scalar field values in parameter space. From this step the range of scalars across the surface can be estimated. Figure 5(b) shows slabs of constant mean curvature for a hyperbolic paraboloid surface, rendered with a constant color.

We can also render curvature as a curvature map where we establish a color gradient and map the range of curvature values on the surface to the color gradient. The range of curvature across the volume being sampled is found by rendering the surface without contours, but calculating the Gaussian or Mean curvature at every point on the surface and saving the maximum and minimum curvature values. This step is necessary so that the range of curvature values across the surface can be calculated. We then map this range of curvature values to colors using the color gradient. Figure 6(a) shows a map of the Gaussian curvature for Steiners surface

$$x^2y^2 + y^2z^2 + x^2z^2 + xyz = 0. \tag{13}$$

Figure 6 (b) shows a map of the Gaussian curvature of the tri-axial tri-torus (3) surface. This surface belongs to the class of non-orientable surfaces and the lines of curvature discontinuity clearly stand out in the figure.

Examples where point based methods have problems occur. As we test whether the surface point (x, y, z) is within $|\varepsilon|$ of f, any thin sections of diameter less than ε will be rendered as thin tubes of diameter ε. The effect is most evident for Steiner's surface where the three double lines $(x = 0, y = 0, z = 0)$ are rendered as tubes of diameter ε (see Figure 6), also rendered by Sederburg [1989]. Decreasing the value of ε so that the width of the tube is a pixel or less reduces this effect.

5 Summary and Discussion

In this work we have shown how to render slabs of constant width on implicit and parametric surfaces where the display primitive is points. The finite thickness of the slabs helps the visualization of the surfaces because their variable projected widths in the images provide visual clues about surface orientation. This is particularly evident in Figure 2. In this work we render curvature maps of surfaces and use

- slabs parallel to the x, y, and z principle planes,
- slabs centered on an axis and rotated by multiples of an arbitrary angle around that axis,
- slabs that are concentric spheres of radius r_i,
- slabs which are iso-values of the Gaussian or mean curvature of the surface,

Using these techniques we can successfully render regular, singular, and non-manifold parametric and implicit surfaces to the pixel precision of the image. The techniques are simple, distributing points on surfaces and rendering them one at a time. The techniques are slower than scan line or ray tracing techniques. However, dealing with single points many standard issues, such as hidden surface removal, clipping, shadow rendering, active edge lists, polygonization are eliminated or reduced to trivial forms. The result is low programming development time.

While our techniques are in general, slower than ray tracing, there are cases where

- ray tracing is impractical or impossible due to the difficulty in solving the ray surface intersection problem in ray tracing,
- the techniques may be faster than ray tracing for complex renderings where the density of surfaces in the ray traced volume is high. Point sampling the volume in this case will probably be more efficient than trying to solve all raysurface intersection equations in that volume.

6 Future Work

Aliasing problems can be seen along the edges of the surfaces and along contour lines. This can be reduced by rendering the image at a higher resolution than that required, at the cost of greater rendering time. However a better scheme of anti-aliasing needs to be developed.

In Balsys and Suffern [2003] we also developed a "thin contour" algorithm that renders one pixel wide contours on ray traced surfaces. This is simpler than the slab algorithm because it doesn't require the gradient of the scalar fields to be calculated. We also modified this algorithm to work with point based rendering, as follows. We test if the x, y, and z coordinates of each point generated on a surface are within $\pm\varepsilon$ of a specified level surface of $g(x, y, z)$. If they are, we shade the point with the contour color; otherwise we use Phong shading. We adjust ε so that contour width is approximately one pixel.

We tested this algorithm with a number of surfaces, and Figure 7 shows the results for a sphere. The contours here have severe aliasing problems. Increasing the value of ε makes the contours wider, but does not significantly reduce the aliasing. We therefore did not pursue this algorithm further; it is not suited to point based rendering.

We use an IFS scheme to stochastically distribute points in the plotting volume. In some cases this means we must generate many points in the volume when only a few pixels are missing from the final image. Surface sections that are more parallel to the viewing direction are filled much more rapidly than surface sections that are more normal to the viewing direction. It may be possible to use this information to tailor the way we generate points to some optimum. As we obtain values of points on (or near) the surface the points are independent of the viewing system and thus it is simple (and fast) to obtain images from varying viewpoints as the points themselves do not need to be regenerated - only the view needs to be redone.

Points generated uniformly in parameter space are not distributed uniformly on parametric surfaces; for example, we see concentrations of more densely packed points at the poles of the surface where iso-parameter curves are more tightly packed. This is not just a fault of the random methods, but a consequence of uniform distribution in the parameter space. Something to be considered for future work is biasing distribution of random variables in the (u, v) parameter space to reduce this effect, perhaps using the first fundamental form for f.

We also need to speed up the process of generating points on the visible parts of the surface, particularly for implicit surfaces. We are currently investigating the use of octrees to speed up the rendering time of implicit surfaces.

References

Balsys, R.J., and Suffern, K.G. 2001. Visualisation of implicit surfaces, Computers and Graphics, 25:89–107.

Balsys, R.J., and Suffern, K.G. 2003. Ray Tracing Surfaces with Contours, Computer Graphics Forum, 22:4:1–10.

Barr, A.H. 1981. Superquadratics and Angle-Preserving Transformations. IEEE CG&A, pp. 11–23, Jan. 1981.

Bloomenthal, J. 1988. Polygonization of implicit surfaces. Computer Aided Geometric Design, 5:341–355.

Bourke, P. 2001. http://astronomy.swin.edu.au/pbourke/geometry/klein/ visited December 2001.

Bourke, P. 2003. http://astronomy.swin.edu.au/ pbourke/surfaces/tritorus/ visited December 2003.

Chandru, V., Dutta, D., Hoffmann, C. M. 1989. On the geometry of Dupin cyclides, The Visual Computer, 5:277-290.

De Figueiredo L. H. and Gomes J. 1996. Sampling implicit surfaces with physically-based particle systems, Computers and Graphics, 20:3:365–375.

Foley, J.D., Van Dam, A., Feiner, S.K., and Hughes, J.F. 1990. Computer graphics: principles and practice (2 ed.), Addison-Wesley, Reading, MA.

Hagen, H., Schreiber, T., Gschwind, E., 1990. Methods for surface interrogation. Visualisation '90 Proceedings, Los Alamitos, IEEE Computer Society Press, 187–193.

Hagen, H., Hahmann, S., Schreiber, T., Nakajima, Y., Wïrdenweber, B., Hollemann, P., 1992. Surface interrogation algorithms. IEEE CG&A, 12(5):53–60.

Hanrahan, P. 1983. Ray tracing algebraic surfaces, Computer Graphics (SIGGRAPH 83 Conference Proceedings), pp. 83–90.

Higashi, M., Kushimoto, T., Hosaka, M., 1990. On formulation and display for visualising features and evaluating quality of free form surfaces. Eurographics '90 Proceedings, North-Holland, Amsterdam, 299–309.

Hoitsma, D.H., Roche, M., 1983. The Computation of All Plane-Surface Intersections for CAD/CAM Applications. Computer Aided Geometry Modeling, NASN C.P. 2272:15–18.

Jones, H. 2001. Exact object rendering through iterated function systems. Eurographics UK Chapter Annual Conference, University College, London. pp. 27–36.

Jones, H., Balsys, R.J., Suffern, K.G. 2003. Point Based Rendering of Surfaces With Singularities. ACM/GRAPHITE2003, 11-14 February 2003, Melbourne, Australia. pp. 267268.

Jones, H. and Moar, M. 2000. Rendering through iterated function systems. In Paradigms of complexity: fractals and structures in the sciences. M. Novak (ed.). World Scientific, Singapore, pp. 167–177.

Lee, R.B., Fredericks, D. A., 1984. Intersections of a Parametric Surface and a Plane. IEEE CG&A, 4(8):48–51.

Levoy, M., Whitted, T. 1985. The Use of Points as a Display Primitive, Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, http://graphics.stanford.edu/papers/points.

Lorenson, W.E., and Cline. H.E. 1987. Marching cubes: a high resolution 3D surface construction algorithm. Computer Graphics, 21:163–170.

Nordstrand, T., 2001. http://www.uib.no/People/nfytn/mathgal.htm, visited December 2001.

Maas, A., 1994. Het simuleren van oppervlak-kwaliteit bij een CAD-model. Literatuuronderzoek IO87, Internal report in Dutch, Delft University of Technology, Delft.

Petersen, C.S., 1984. Adaptive Contouring of Three-Dimensional Surfaces. Computer Aided Geom. Design, 1(1):61–74.

Rockwood, A. 1987. A Generalised Scanning Technique for Display of Parametrically Defined Surfaces, IEEE Computer Graphics and Applications, 7:8:15–26.

Rösche, A., Ruhle, M., Saupe, D. 1997. Interactive Visualization of Implicit Surfaces with Singularities, Computer Graphics Forum, 16:5:295–306.

Satterfield, S.G., Rogers, D.F., 1985. A Procedure for Generating Contour Lines from a β-Spline Surface. IEEE CG&A, 5(4):71–75.

Schmidt, M.F.W. 1990. Cutting cubes - visualizing implicit surfaces by adaptive polygonization. The Visual Computer, 10:101–115.

Sederburg, T.W., and ZundeL, A.K. 1989. Scan Line Display of Algebraic Surfaces. Computer Graphics (SIGGRAPH 89 Conference Proceedings), pp. 147–156.

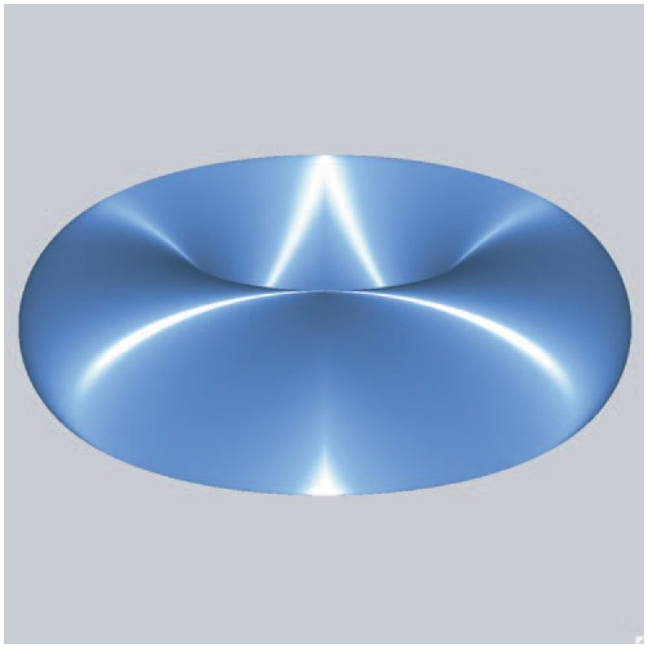
Spivac, M., 1979. A Comprehensive Introduction to Differential Geometry, Second Edition, Publish or Perish, Berkeley, Vol. 3, pp. 204.

Tanaka S., Morisaki A., Nakata S, Fukuda Y., and Yamamoto H. 2000. Sampling implicit surfaces based on stochastic differential equations with converging constraint, Computers & Graphics, 24:419-431.

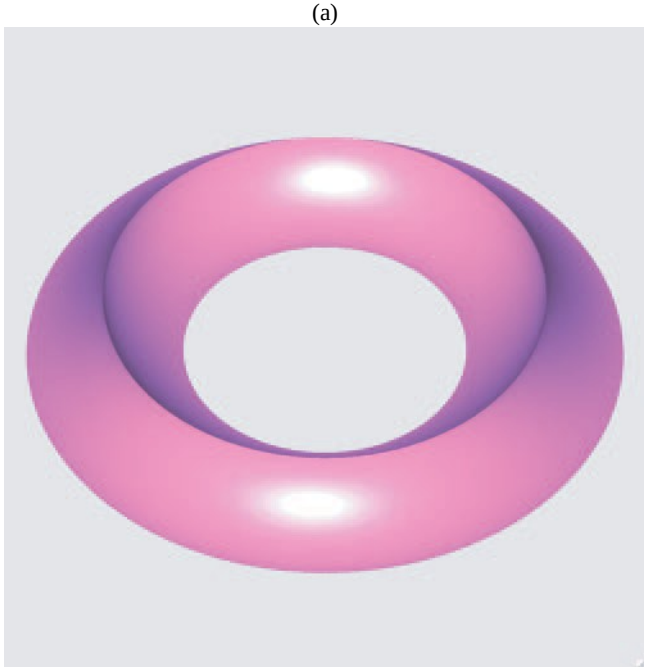
Tanaka, S., Shibata, A., Yamamoto, H., Kotsuru, H. 2001. Generalized Stochastic Sampling Method for Visualization and Investigating of Implicit Surfaces, Proc. Eurographics 2001, Computer Graphics Forum, 20:3:359–367.

Witkin A. P., and Heckbird P. S. 1994. Using Particles to Sample and Control Implicit Surfaces. Proceedings of SIGGRAPH 94. In Computer Graphics Proceedings, Annual Conference Series, pp. 269–277.

Wyvill, C.G., McPheeters, C., WyvilL, B. 1986. Data structures for soft objects. The Visual Computer, 2:227–234.

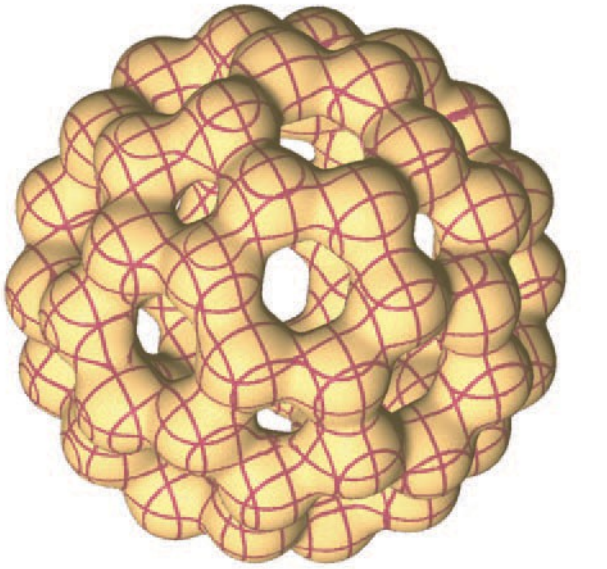


(a)

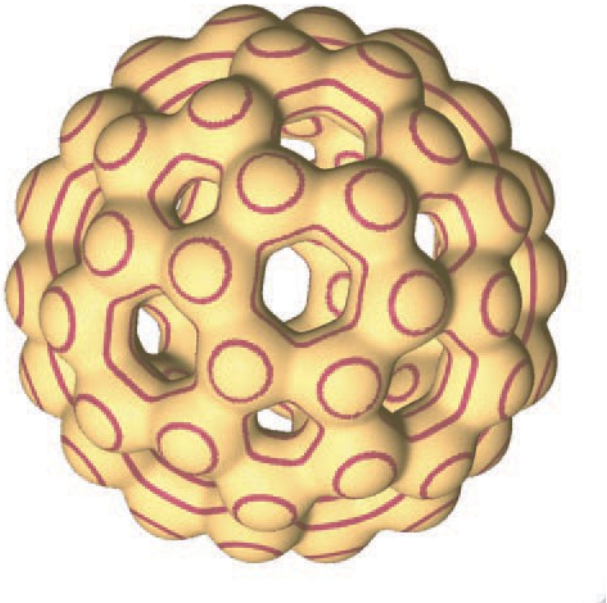


(b)

Figure 1: (a) Points on surface of the parametric triaxial tri-torus (3) and (b) Points on surface of an implicitly defined super-toroid (5) surface with $a = b = c = 5, d = 20, \varepsilon_1 = 3$ and $\varepsilon_2 = 1$.

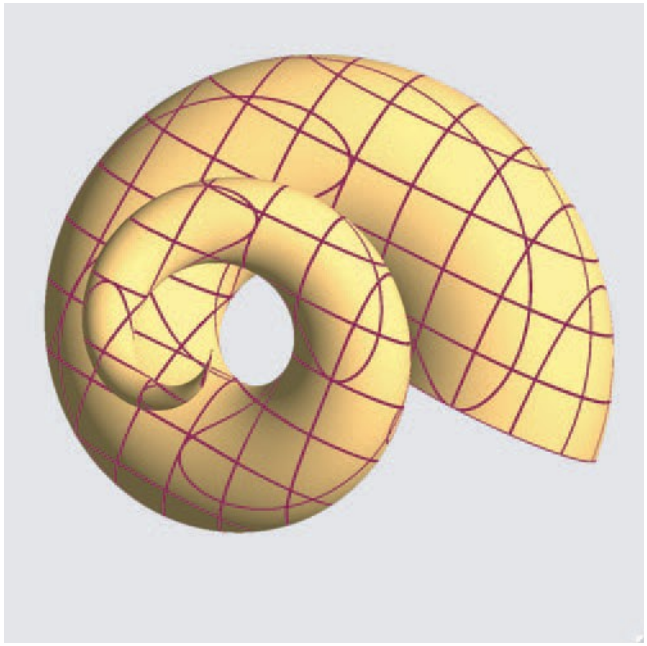


(a)

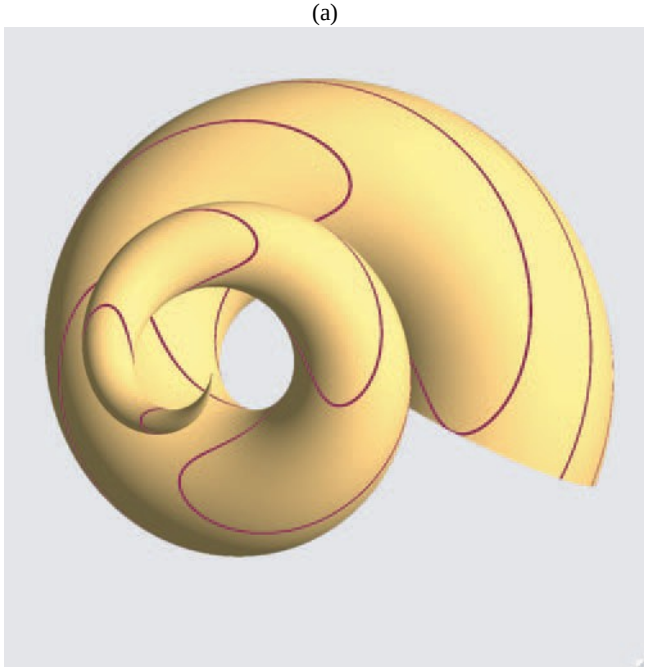


(b)

Figure 2: The Buckyball surface (8) with 60 Carbon atoms on the surface of a geodesic sphere (a) with slabs parallel to the principal planes and (b) with slabs that are concentric spheres of radius r_i.

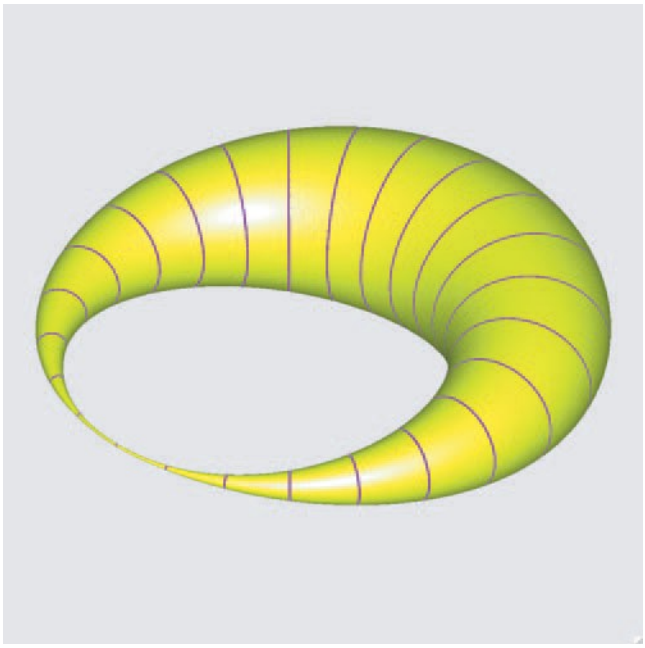


(a)



(b)

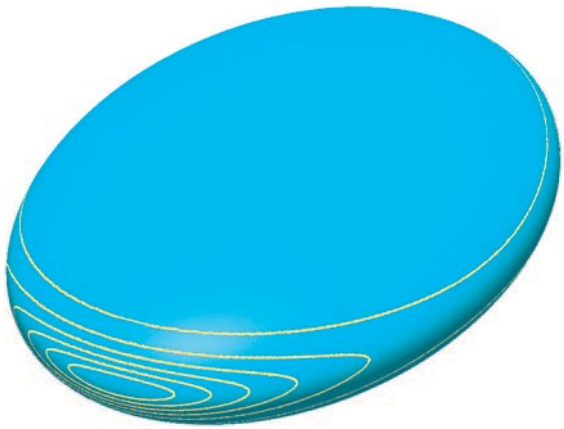
Figure 3: The shell or ram's horn surface (9) (a) with slabs parallel to the principal planes and (b) with only slabs parallel to the z principle plane.



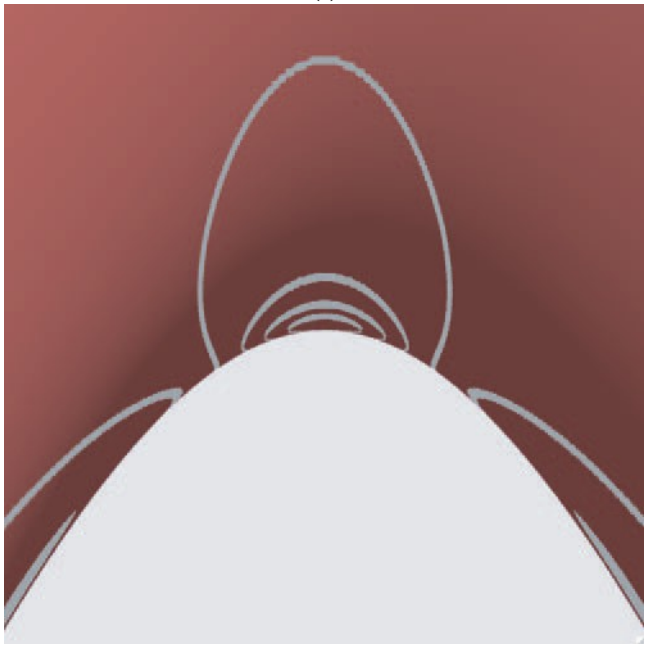
(a)

(b)

Figure 4: (a) The Cyclide surface (10) with slabs lying along the z axis and rotated by multiples of 15° around the z axis and (b) the Klein Bottle surface (11) with $0 \leq v \leq \pi$ and slabs lying along the z axis and rotated by multiples of 15° around the z axis.

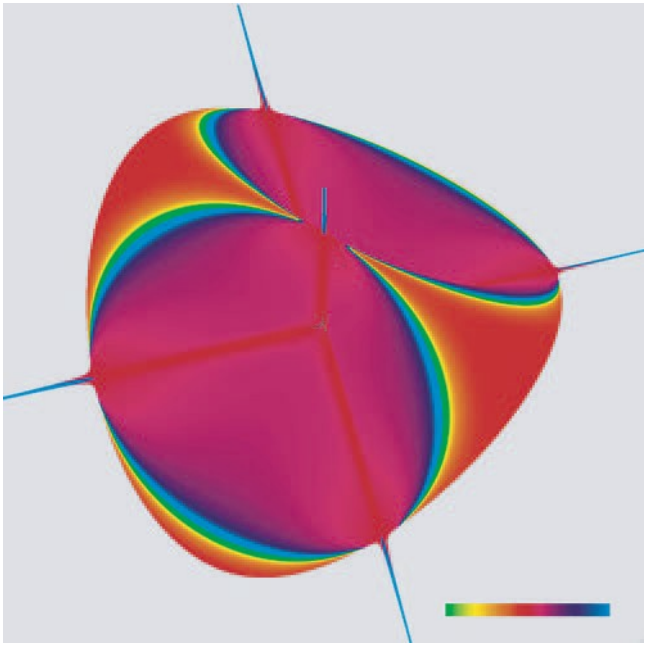


(a)

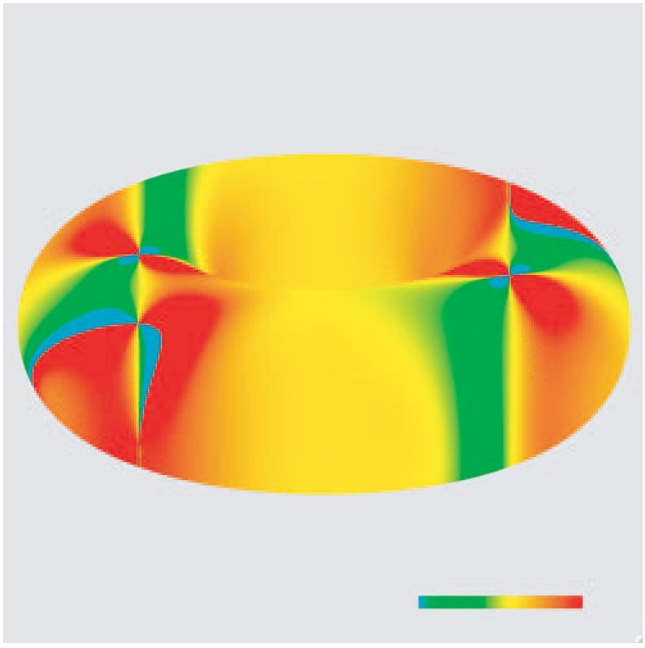


(b)

Figure 5: (a) Gaussian curvature slabs of an ellipsoid (12) and (b) the Mean curvature slabs on the hyperbolic paraboloid surface $f(x, y, z) = -(\frac{x}{a})^2 + (\frac{y}{b})^2 + z$.



(a)



(b)

Figure 6: (a) A Gaussian curvature map of steiners surface (13), legend lower right goes from minimum Gaussian curvature (green) to maximum Gaussian curvature (cyan). (b) A Gaussian curvature map of the tri-axial tri-torus (3).

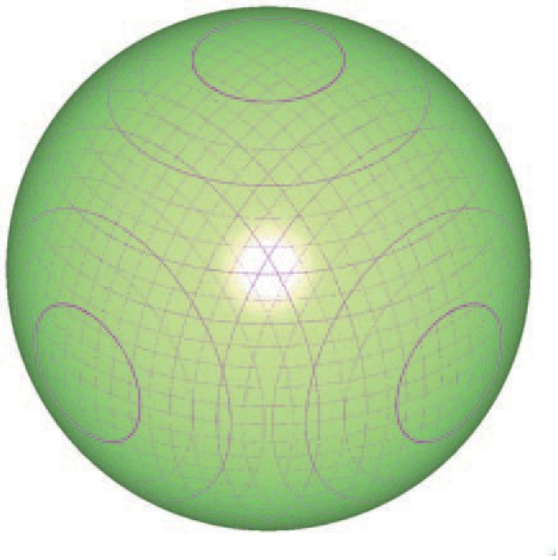


Figure 7: A sphere rendered with thin contours showing aliasing problems.