Polygonization of Multi-Component Non-Manifold Implicit Surfaces through A Symbolic-Numerical Continuation Algorithm

Adriano N. Raposo Abel J. P. Gomes

Universidade da Beira Interior
Departamento de Informática, Instituto de Telecomunicações
6200-001 Covilhã, Portugal

Figure 1: $(x^2+y^2+z^2-2)^2 \cdot (\sin(x)+y+1) = 0$.

Abstract

In computer graphics, most algorithms for sampling implicit surfaces use a 2-points numerical method. If the surface-describing function evaluates positive at the first point and negative at the second one, we can say that the surface is located somewhere between them. Surfaces detected this way are called sign-variant implicit surfaces. However, 2-points numerical methods may fail to detect and sample the surface because the functions of many implicit surfaces evaluate either positive or negative everywhere around them. These surfaces are here called sign-invariant implicit surfaces. In this paper, instead of using a 2-points numerical method, we use a 1-point numerical method to guarantee that our algorithm detects and samples both sign-variant and sign-invariant surface components or branches correctly. This algorithm follows a continuation approach to tessellate implicit surfaces, so that it applies symbolic factorization to decompose the function expression into symbolic components, sampling then each symbolic function component separately. This ensures that our algorithm detects, samples, and triangulates most components of implicit surfaces.

CR Categories: I.3 [COMPUTER GRAPHICS]: ;— [I.3.5]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations.

Keywords: Implicit surfaces, polygonization, symbolic factorization, numerical methods.

1 Introduction

Implicit surfaces are widely used in computer graphics and visualization. An implicit surface $\mathscr{S}$ is a level set (or zero set) of some function f from $\mathbb{R}^n$ to $\mathbb{R}$, say $\mathscr{S} = \{x \in \mathbb{R}^n : f(x) = 0\}$. In this paper, we consider the problem of representing 3D implicit surfaces defined by real functions. In other words, we aim at computing a polygonal approximation for a surface $\mathscr{S}$ defined implicitly by $f : \Omega \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$, i.e. $\mathscr{S} = \{(x,y,z) \in \mathbb{R}^3 : f(x,y,z) = 0\}$.

Basically, there are three categories of algorithms to render implicit surfaces, namely:

- *Spatial sampling techniques*. These algorithms subdivide (either regularly or adaptively) the space into a lattice of cells to find those that intersect the implicit surface [Bloomenthal 1988] [Hall and Warren 1990] [Lorensen and Cline 1987] [Muller and Stark 1993] [Stander and Hart 1997] [Velho et al. 1999]. Usually, cells are either cubes or tetrahedra. The sign of the surface-describing function at the cell vertices determines a configuration type that guides the ongoing polygonization. Unlike cubes, tetrahedra generate topologically consistent triangular meshes (i.e. without ambiguities), yet with distorted triangles. These distorted triangles require some kind of post-processing procedure to repair the resulting mesh. Even worse, the cubic cell-based polygonization leads to ambiguous configurations because more than one mesh may be created for the same configuration type. Some disambiguation strategies have been proposed in the literature, including simplex decomposition, modified look-up table disambiguation, gradient consistency heuristics and quadratic fit, tri-linear interpolation techniques, and recursive subdivision of space into smaller sub-cells.
- *Surface tracking techniques*. Also known as *continuation methods*, they iteratively create a polygonal approximation of the surface by using a mesh growing scheme from a starting seed element on the surface [Dobkin et al. 1990] [Melville and Mackey 1995]. This seed element results from the intersection between the seed cell and the surface. Neighbor surface elements are found by intersecting the neighbor cells with the surface, i.e. the mesh growth results from the growing of cells straddling the surface. Unfortunately, the algorithm may miss important shape details of the surface, including very small components of the surface or even isolated points, because cell size is constant. Another drawback of this technique comes from the need of having a seed cell for each surface component, which may be a rather difficult requirement to satisfy. Still, other tracking techniques for tilling implicit surfaces include: predictor-corrector (PC) and piecewise-linear (PL) techniques [Allgower and Gnutzmann 1987] [Rheinboldt and Burkardt 1983] [Rheinboldt 1987] [Brodzik 1998].

GRAPHITE 2006, Kuala Lumpur, Malaysia, November 29 – December 02, 2006.
© 2006 ACM 1-59593-564-9/06/0011 $5.00

- *Surface fitting techniques*. Unlike spatial sampling techniques, surface fitting methods are not based on partitioning space into cells. Starting from a seed mesh that roughly approximates the implicit surface, these techniques progressively adapt and deform the current mesh to the implicit surface [Schmitt et al. 1986] [Zhou et al. 1997] [Li et al. 2004]. The main problem comes from the difficulty in attaching triangle patches together, which sometimes results in cracks or even dangling triangles in the tessellation.

Our algorithm belongs to second category: *continuation methods*. It tessellates the surface progressively from a seeding point that works as the center of a first hexagon of triangles. The vertices of each triangle are determined by a surface sampling process based on a 1-estimate numerical algorithm, in particular a 3D Newton Predictor-Corrector numerical algorithm. Although it normally requires significant computation, triangulation followed by triangle rendering is more efficient than direct rendering methods such as volume visualization or ray tracing [Bloomenthal 1997].

The rest of this paper is organized as follows. Section 2 provides us an overview of our algorithm. Section 3 shows us the importance of the factorization techniques applied to multi-component implicit surfaces. Section 4 describes how an implicit surface is sampled through the Newton Predictor-Corrector. Section 5 describes the surface triangulation based on the progressive mesh growth. Section 6 describes some experiments we have carried out to validate our algorithm. Section 7 comes up with the difficulties of current continuation methods in dealing with singularities, making a comparison between our algorithm and others found in the literature. Finally, in Section 8 we draw some relevant conclusions.

2 Algorithm Overview

As said before, our algorithm is in the class of *continuation algorithms*. The main drawback of these algorithms is that we do not know *a priori* the topology (or topological shape) of the surface to be tessellated. This means that we do not know how many topological components it possesses. Consequently, it is rather difficult to devise a solution to find a seeding point in each surface component in order to successfully polygonize the whole surface.

Our proposal to solve the previous problem is to proceed to factorization of the function into function components, also called *symbolic components* or *irreducible components*, before the surface sampling. A function f is called irreducible if it is non-constant and cannot be represented as the product of two or more non-constant function components. Every function f can be factorized into irreducible function components. This factorization is unique up to permutation of the factors and the multiplication of constants. In this context, an irreducible implicit surface is a surface represented by an irreducible implicit function.

Let f be the implicit surface function and let $f_1, \ldots, f_n$ be irreducible functions resulting from the symbolic factorization of f, i.e., $f = f_1 \cdot \ldots \cdot f_n$. We consider that each f_i is a function that represents one of the symbolic components of the main surface. The overall algorithm can be then described as follows:

1. Factorization of f into irreducible function components f_i.
2. For each f_i apply the proposed sampling and triangulation algorithm.
3. Determination of the curves of intersection between irreducible components f_i.
4. Render the surface, i.e. its triangles (using Java3D/OpenGL).

After sampling and triangulating all irreducible components of a surface, it is necessary to determine their intersection curves. This can be done easily by determining intersecting triangles of the irreducible components. For example, the surface $((x^2+y^2+z^2-2)^2) \cdot (\sin(x)+y+1)$ shown in Figure 9 (d) has two irreducible components that intersect each other. We use the algorithm proposed by Möller [Möller 1997] to determine intersection curves between irreducible components of an implicit surface. This procedure is not necessary for visualization purposes, but it is essential for representing non-manifold surface meshes correctly into a geometric data structure.

3 Function Factorization

Following the idea of decomposing a problem into more tractable subproblems, we use factorization techniques to decompose a function expression into subexpressions (or symbolic components). This is important because tessellating each symbolic component separately is *a priori* simpler than tessellating the surface as a whole.

There are three types of symbolic components, namely:

1. *One symbolic component embodies a topological component*. Many times, a symbolic component corresponds to a topological component. For example, the surface depicted in Figure 9(e) has two symbolic components, $(x^2+y^2+z^2-2)^2=0$ and $(x^2+(y+3)+z^2-1)^2=0$, each corresponding to a topological component, a sphere and a paraboloid, respectively. That is, we know *a priori* the overall topology of the surface, i.e. the number of their topological components. This allows us to sample and polygonize each surface component surrounding the main drawback of continuation algorithms.
2. *Two or more symbolic components embody a topological component*. This is the case of the surfaces shown in Figure 9(a), (c), and (d). In this case there is intersection between the symbolic components, being each symbolic component tessellated independently of the others. The separate tessellation of each symbolic component ensures that intersection curves need not be computed symbolically or have a particular procedure to treat these surface self-intersections during triangulation stage. It is enough to apply the Möller's triangle-to-triangle algorithm to polylinearize the intersection curves after triangulating all symbolic components [Möller 1997].
3. *One symbolic component has two or more topological components*. In this case, our algorithm only detects and tessellates one topological component of the target symbolic component. For example, the hyperboloid of two sheets $-x^2-y^2+z^2=1$ has a single symbolic component with two topological components. Taking into account that we use a single seeding point for each symbolic component, then only one topological component can be tessellated. To overcome this problem we need to know *a priori* the topological type of the symbolic component, i.e. its number of topological components. Recent results about the computation of the topological type of implicit curves and surfaces will be certainly useful to solve this problem in the near future [Gonzalez-Vega and Necula 2002] [Seidel and Wolpert 2005] [Mourrain and Técourt 2005].

Thus, factorization seems to be the appropriate technique to correctly polygonize multi-component implicit surfaces through a continuation method. Possibly, this is one of the main contributions of this paper.

4 Surface Sampling

After decomposing a surface into its symbolic components or irreducible surfaces, we proceed to sample and triangulate each irreducible surface. Sampling a symbolic component is carried out by means of a Predictor-Corrector algorithm.

4.1 Predictor: Points on the Tangent Plane

The leading idea of triangulating a surface here is to generate a nearly regular mesh. Thus, in general, every sampled point of the surface will be sooner or later the center of a nearly regular hexagon (Figure 2). Let us then consider an *arbitrary* point inside the considered bounding box. Applying the Newton Corrector to such a point we obtain the point p_0 on the surface after some iterations. The star of the hexagon vertices $q_1, q_2, \ldots, q_6$ are first predicted on the plane that is tangent to the surface at p_0. Then, each predicted point q_i is corrected towards a point p_i belonging to the surface. This correction is done through the 3-dimensional counterpart of Newton-Raphson's root-finding algorithm, as described in the next subsection. The calculation of the hexagon vertices may be either total or partial. It is total when p_0 is the center of the former hexagon. On the other hand, some hexagon vertices surrounding p_0 may already exist in the surface-triangulating mesh so that we need only to determine the remaining hexagon vertices around p_0 (Figures 5 and 6). In this case, the calculation of the hexagon vertices is said to be partial.

4.2 Newton Corrector: Surface as Attractor of Points

In numerical analysis, the iteration formula has the following generic form

$$x_{k+1} = F_k(x_k, x_{k-1}, \ldots, x_{k-n+1}). \tag{1}$$

This is called an *n-points iteration function*. Most surface polygonizers use 2-points iteration functions, i.e. polarity-based polygonizers. They use two estimates x_k and x_{k-1} to compute the next one x_{k+1}. The *false position method* (or *regula falsi*) and the *secant method* are two examples of numerical methods that use 2-points iteration functions. In this case, we say that a root is bracketed in the interval defined by two points x_k and x_{k-1} if $f(x_k)$ and $f(x_{k-1})$ have different signs, where f is the function that represents the implicit surface. This is so because, according to the Intermediate Value Theorem, there must be at least one root in $[x_{k-1}, x_k]$, unless a singularity is present. The well-known algorithm of the marching cubes is a sort of algorithm that uses a 2-points iteration function for each pair of vertices bounding each cubic cell edge of the partitioning space.

However, these 2-points iteration functions are not able to detect sign-invariant branches and components of an implicit surface, simply because its function does not change sign in the neighborhood of each one of their points. For example, the spherical surface $(x^2+y^2+z^2-9)^2 = 0$ cannot be sampled by the traditional 2-points iteration functions because the surface-describing function is positive everywhere, unless on the surface itself where it is zero.

Instead, we use the 3-dimensional counterpart of classical Newton method, which is polarity-independent. It uses 1-point iteration function that is given by

$$F_k(\mathbf{p}_k) = \mathbf{p}_k - \frac{f(\mathbf{p}_k)}{\nabla f(\mathbf{p}_k)^2} \nabla f(\mathbf{p}_k). \tag{2}$$

where $\nabla f(\mathbf{p}_k)$ is the gradient of f at $\mathbf{p}_k \in \mathbb{R}^3$. It produces successive estimates $\mathbf{p}_{k+1}$ that converges to a surface point $\mathbf{p}$ from a first guess $\mathbf{p_0} = \mathbf{q}$. This iterative process stops when $||\mathbf{p}_{k+1} - \mathbf{p}_k|| \leq \varepsilon$ is sufficiently small, i.e. $\mathbf{p} \approx \mathbf{p}_{k+1}$.

Recall that the Newton method has quadratic convergence, though it may not converge when the function oscillates rapidly. In the case of implicit surfaces, the divergence phenomenon is rare, even when there are significant curvature variations locally. Obviously, nobody expects that the algorithm works well for pathological surfaces like $f(x,y,z) = \sin(\frac{1}{x})$. Nevertheless, given a symbolic component of the surface and any point in the pre-defined bounding box, it is always possible to find a surface point of such a symbolic component by applying the formula (2) after some iterations. This means that such a symbolic component works as an attractor for any points in the bounding box, in particular for those on the tangent plane to a surface at a point. Note that there is here a subtle fact when we say that given a point in a bounding box it always (apart from pathological surfaces) converges to the current symbolic component, not to other part of the surface.

5 Surface Triangulation

Paving a surface with approximately regular hexagons (partitioned into six triangles) comes from the common idea that the hexagon is the best geometric figure to tessellate a surface. Tessellating a surface means here to tessellate each of its symbolic components.

As usual, polygonizing each symbolic component of the surface requires a pre-definition of a bounding box where it lies in totally or partially. Taking in account that a symbolic component may possess one or more boundaries on the bounding box, its corresponding mesh ends up by having one or more boundaries as well. But, at the beginning of the triangulation, we have only a single mesh boundary that is just the boundary of the first hexagon. Thus, we need a way of splitting mesh boundaries (Figure 7). Besides, there may be a need for merging mesh boundaries when they get very close to each other (Figure 8).

5.1 Starting hexagon

As for any other *continuation algorithm*, the polygonization starts with a seeding point. Let p be a *arbitrary* point in the bounding box containing a surface's symbolic component or part of it. The first sampling point p_0 of the symbolic component is calculated by applying the Newton Corrector to p. It is necessary to compute and store the vectors $\overrightarrow{n}$ (the normal vector to the surface at p_0), $\overrightarrow{t_1}$ and $\overrightarrow{t_2}$, in order to construct an orthonormal basis at p_0. After this, let α be the tangent plane to the symbolic component at the point p_0. Considering a circle with radius δ on α centered in p_0, we calculate the points $q_1, \ldots, q_6$ as follows:

$$q_{i+1} = p_0 + \delta \cos(i\pi/3)\overrightarrow{t_1} + \delta \sin(i\pi/3)\overrightarrow{t_2} \tag{3}$$

Note that the points q_i are the vertices of a regular hexagon inscribed in a disk centered at p_0 having six equilateral triangles inside. The disk is placed on the plane tangent to the surface at p_0. To calculate the corresponding six sampling points on the surface we apply the Newton Corrector to the points q_i. The resulting points

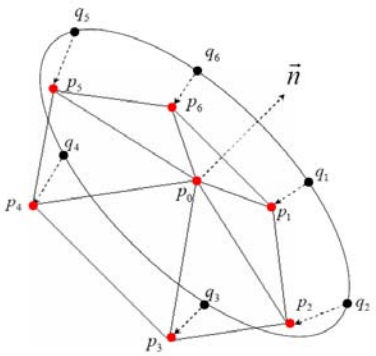

Figure 2: Starting hexagon.

$p_1, \ldots, p_6$ are the six vertices of the starting hexagon of the surface, that is, the first mesh boundary. This is illustrated in Figure 2.

5.2 External angle at a mesh boundary vertex

The mesh expansion or growth is determined by the external angle at each vertex v_i of each mesh boundary. Figure 3 illustrates the concept of external angle θ at a vertex v_i, as well as the corresponding internal angle $\phi = 2\pi - \theta$. Note that the external angle θ is not necessarily the angle between the vectors $\overrightarrow{u} = \overrightarrow{v_i v_{i-1}}$ and $\overrightarrow{v} = \overrightarrow{v_i v_{i+1}}$. Recall that, by definition, $\angle(\overrightarrow{u}, \overrightarrow{v})$ yields the minimum angle between $\overrightarrow{u}$ and $\overrightarrow{v}$. There are cases, like in Figure 3, where the external angle θ is not equal to $\angle(\overrightarrow{u}, \overrightarrow{v})$; it is precisely $2\pi - \angle(\overrightarrow{u}, \overrightarrow{v})$. To correctly determine the external angle at v_i, we have to check whether the normal vector $\overrightarrow{n}$ at v_i and $\overrightarrow{t} = \overrightarrow{u} \times \overrightarrow{v}$ have opposite signs or not. If $\overrightarrow{t}.\overrightarrow{n} > 0$, then the external angle $\theta = \angle(\overrightarrow{u}, \overrightarrow{v})$; otherwise, $\theta = 2\pi - \angle(\overrightarrow{u}, \overrightarrow{v})$. The correct computation of the external angle at a mesh boundary vertex prevents the re-meshing backwards.

5.3 Approximately uniform partition of the minimum external angle

Let $\Lambda_0 = \{v_1, v_2, \ldots, v_n\}$ be the current mesh boundary with external angles $\theta_1, \theta_2, \ldots, \theta_n$, respectively. The mesh expansion occurs around the vertex at which the external angle is minimum. Once found the minimum external angle θ, it must be partitioned into a set of angles with approximately $\frac{\pi}{3}$ radians, provided that we are constructing approximately regular hexagons and equilateral triangles.

In this step we want to find the optimal number of triangles that fit θ. Thus, we consider a range of acceptable angles around the ideal angle $\frac{\pi}{3}$ that goes from θ_{inf} to θ_{sup}. These two bounds result, respectively, from the addition and subtraction of a pre-defined tolerance to and from $\frac{\pi}{3}$. Next, it is necessary to calculate the numbers of triangles n_{inf} and n_{sup} that result from dividing θ by θ_{inf} and θ_{sup}, respectively, rounding them to the nearest integer afterwards. Then, we choose either n_{inf} or n_{sup} as the optimal number n_Δ of triangles, i.e. triangles whose angles are closest to $\frac{\pi}{3}$:

$$n_\Delta = \begin{cases} n_{inf} & \text{if } |\frac{\theta}{n_{inf}} - \frac{\pi}{3}| \leq |\frac{\theta}{n_{sup}} - \frac{\pi}{3}| \\ n_{sup} & \text{if } |\frac{\theta}{n_{inf}} - \frac{\pi}{3}| > |\frac{\theta}{n_{sup}} - \frac{\pi}{3}| \end{cases}$$

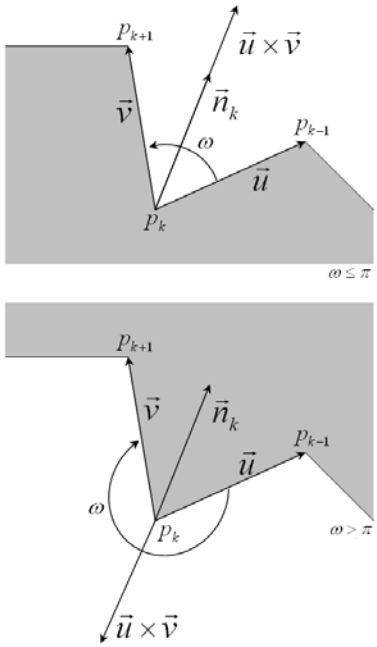

Figure 3: External angle.

As an example, the angle illustrated in Figure 4 would be partitioned in 3 triangles, i.e., $n_t = 3$, because this partition creates angles closest to $\frac{\pi}{3}$.

Sometimes, it happens that n_Δ evaluates to 0. It may also happen that the distance between the previous vertex v_{i-1} and next vertex v_{i+1} is less than the circle radius δ on the tangent plane to the surface at v_i(see 5.1). It is clear that n_Δ cannot be equal to 0, so it is set to 1.

5.4 Mesh growth

Once calculated the number of triangles that fit θ around v_i, the current mesh is ready to grow. The mesh grows as follows:

- *Attaching only one triangle*. This case occurs when $n_\Delta = 1$. For that, as illustrated in Figure 5 it is enough to connect v_{i-1} to v_{v+1} by a new edge to form the triangle defined by v_{i-1}, v_i and v_{v+1}.

- *Attaching two or more triangles*. In this case, $n_\Delta \geq 2$, the mesh grows by attaching two or more triangles around v_i, as illustrated in Figure 6 (grey triangles represent previously polygonized mesh). For that, one first determines the angle $\alpha = \frac{\theta}{n_\Delta}$ of each slice triangle. Next, one computes the point that results from the orthogonal projection of v_{i-1} on a plane tangent to the surface at v_i. Then, one rotates the projected point by α radians about an axis perpendicular to the surface at the point v_i. This way, we obtain a new point on the tangent plane that is attracted by the surface by using the Newton Corrector. The result is a new sampled surface point and a new triangle for the surface mesh. This procedure is repeated $n_\Delta - 2$ times, viewing that the last triangle is attached according to first case above.

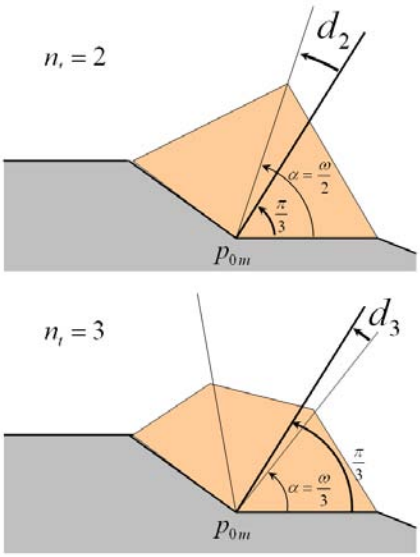

Figure 4: Approximately uniform angle partition.

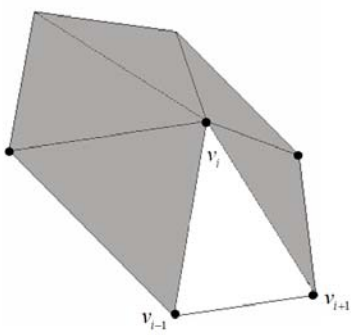

Figure 5: Attaching only one triangle.

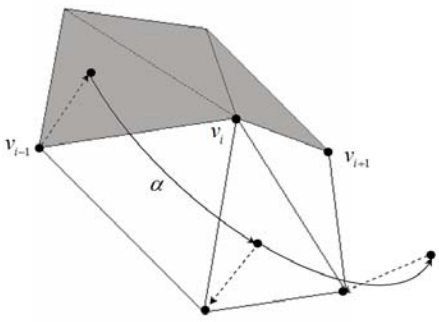

Figure 6: Attaching two or more triangles.

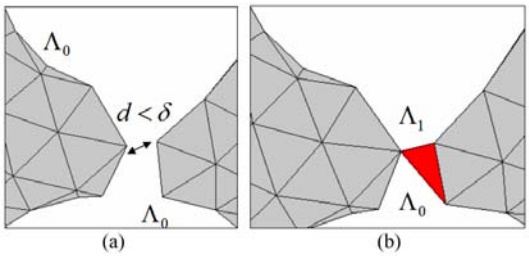

Figure 7: Two non-consecutive vertices belonging to the same mesh boundary are near to each other.

5.5 Mesh overlapping

As in many *continuation*-based algorithms it is necessary to prevent that the mesh overlaps itself, i.e. to re-polygonize surface regions. This control must be performed immediately before the mesh growth, being based on two proximity criteria:

- *Two non-consecutive vertices of the same mesh boundary are near to each other.* If the distance d between two non-consecutive vertices belonging to the same expansion boundary is less that the radius δ of the disk used in the mesh growth step, then they must be connected by a new edge of a new triangle, which splits their expansion boundary into two. This *boundary splitting* operation is illustrated in Figure 7. (The existence of thin triangles in Figure 7, when triangle regularity is claimed, can be justified by the cut done by the bounding box faces).

  Let Λ_0 be a mesh boundary, and let v_i and v_j (with $i < j$) be two vertices of Λ_0 having at least two vertices of Λ_0 in between. If the Euclidean distance between v_i and v_j is less than δ (see Figure 7 (a)), Λ_0 splits into two new mesh boundaries, Λ_0 itself and Λ_m. Thus, after splitting the former Λ_0, the new Λ_0 is given by $\{v_1, \ldots, v_i, v_j, \ldots, v_N\}$ (where N is the number of vertices of the former Λ_0) and Λ_m consists of $\{v_i, \ldots, v_j\}$.

- *Two vertices belonging to distinct mesh boundaries are near to each other.* In this case, two vertices are within a distance that is less than δ, but they do not belong to the same expansion boundary. Consequently, their boundaries are merged into a single one. This *boundary merging* operation is illustrated in Figure 8.

  Let Λ_0 and Λ_m two expansion boundaries. If there is a vertex $v_{0i} \in \Lambda_0$ and another vertex $v_{mj} \in \Lambda_m$ such that the Euclidean distance between them is less than δ, the boundaries are going to be merged into each other (Figure 8). Thus, Λ_m is eliminated and his vertices are transferred into Λ_0, i.e.

  $$\Lambda_0 = \{v_{01}, \ldots, v_{0i}, v_{mj}, \ldots, v_{mN_m}, v_{m1}, \ldots, v_{mj}, v_{0i}, \ldots, v_{0N_0}\},$$

  where N_0 and N_m are, respectively, the numbers of vertices of the former Λ_0 and Λ_m.

6 Development tools and experiments

The experiments were carried out on a laptop computer with Intel Pentium 4 processor at 2.4 GHz, 736 MB RAM and ATI RADEON IGP 340M graphical card with 32 MB of memory, running the Microsoft Windows XP Professional operating system with the Java Sun J2SE 5.0 virtual machine. Our prototype for implicit surfaces, called iLicitAces (*i*mp*Licit* Surf*aces*),

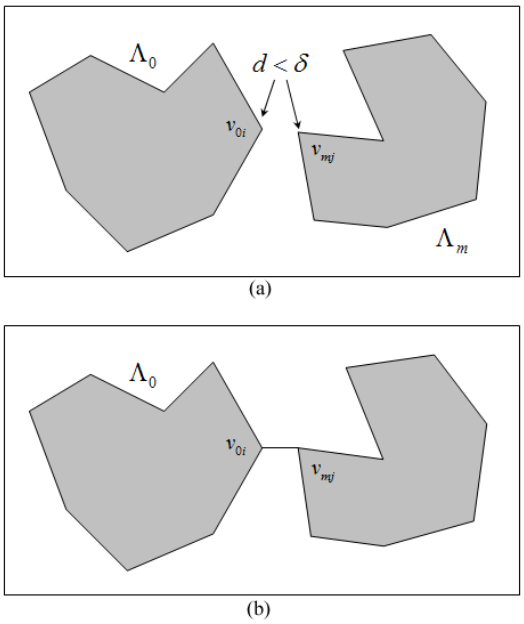

Figure 8: Two vertices belonging to distinct mesh boundaries are near to each other.

has been encoded using several ready-to-use Java APIs to shorten its implementation time. In addition to Java programming language, Java3D API for graphics, and Swing API for user interface, three symbolic computing libraries were also used for parsing, differentiation, and factorization of functions, namely: *JEP* (Java Expression Parser, http://www.singularsys.com/jep), *DJEP* (http://www.singularsys.com/jep/doc/html/djep), and JSCL (Java Symbolic Computing Library, http://jscl-meditor.sourceforge.net).

For all experiments, we used the seeding point $p_0 = (1,1,1)$, the axis-aligned cubic bounding box centered at $(0,0,0)$ with length equal to 6, the hexagon-inscribing disk radius $\delta = 0.3$, and floating-point precision $\varepsilon = 0.001$. Note that the algorithm does not depend on the position of the seeding point. It can be any point inside the bounding box. This is so because of the attractor effect of the surface on the estimates computed by the Newton corrector.

During our experiments, we have considered algebraic and transcendental surfaces, i.e. surfaces defined by polynomials and transcendental functions, respectively, independently of whether they were reducible by factorization or not. Let us enumerate some of them:

1. $f(x,y,z) = x^2 - y^2 = 0$
   (Time: 6.739 sec; Figure 9(a))
   (1st component $x - y = 0$: 1746 triangles)
   (2nd component $x + y = 0$: 1731 triangles)
   This is an algebraic surface with two intersecting symbolic components $x - y$ and $x + y$, i.e. two planes. They both form a topological component.

2. $f(x,y,z) = z - \frac{1}{(x^2+y^2)} = 0$
   (Time: 6.469 sec; Figure 9(b))
   (Number of triangles: 1472)
   This rational surface also possesses a single topological component (Figure 9(b)), described by only one symbolic component. We did not use any gcd (greatest common divisor) symbolic computation here to transform this rational function into an algebraic function.

3. $f(x,y,z) = x\ln x + \ln x \cos z - xy - y\cos z = 0$
   (Time: 6.069 sec; Figure 9(c))
   (1st component $\ln x - y = 0$: 1143 triangles)
   (2nd component $\cos z + x = 0$: 1546 triangles)
   This is a surface with two intersecting symbolic components, $\ln x - y = 0$ and $\cos z + x = 0$, obtained after factorizing the expression of f.

4. $f(x,y,z) = (x^2 + y^2 + z^2 - 2)^2 \cdot (\sin(x) + y + 1) = 0$
   (Time: 7.871 sec; Figure 9(d))
   (1st component $(x^2 + y^2 + z^2 - 2)^2 = 0$: 856 triangles)
   (2nd component $\sin(x) + y + 1) = 0$: 1468 triangles)
   This surface has two intersecting symbolic components. The second component is transcendental. The first is a sign-invariant algebraic component $(x^2 + y^2 + z^2 - 2)^2 = 0$ (i.e. a sphere). In fact, the value of its exponent is 2, so it is positive everywhere, except on the surface where it is zero. Thus, this component cannot be sampled by a conventional 2-points numerical method (e.g. secant method).

5. $f(x,y,z) = ((x^2 + y^2 + z^2 - 2)^2)((x^2 + (y+3) + z^2 - 1)^2) = 0$
   (time: 5.167 sec; Figure 9(e))
   (1st component $(x^2 + y^2 + z^2 - 2)^2 = 0$: 856 triangles)
   (2nd component $(x^2 + (y+3) + z^2 - 1)^2 = 0$: 208 triangles)
   This surface also has two intersecting symbolic components, though they are both algebraic. The first is a sign-invariant sphere $(x^2 + y^2 + z^2 - 2)^2 = 0$. The second is a sign-invariant paraboloid $(x^2 + (y+3) + z^2 - 1)^2 = 0$. Again, these components cannot be detected and sampled by a conventional 2-points numerical method.

These examples show us that symbolic components of a surface can be triangulated separately, so surrounding the difficult problem of triangulating surface with self-intersections.

These examples also show us that our algorithm is capable of sampling and triangulating sign-invariant components. This is due to the fact that the Newton-Raphson is a 1-point numerical method. That is, the surface works as an attractor of the first predicted or estimated point, making the subsequent estimates to converge towards the surface.

As known, the Newton-Raphson method has quadratic convergence, which explains the reasonable rendering times, even taking into account that iLicitAces has been developed using Java technologies. The performance of the prototype could be much better if we had used a compiled programming language, like C++, instead.

7 Comparison to other Continuation Algorithms

Polygonizing and rendering *multi-component implicit surfaces* is straightforward through spatial decomposition of the bounding box into cubic cells. However, continuation methods do not cope well with multi-component surfaces because it is necessary to find a point on each component to start tessellating it. At the best knowledge of the authors, the algorithm here described is the first continuation algorithm to do so.

As far as *self-intersections* are concerned, similar continuation algorithms cannot cope with them. They simply ignore this important issue. On the contrary, we have solved this problem though not fully. In fact, we are able to handle self-intersections that result from intersecting symbolic components, using for that the triangle-to-triangle intersection algorithm due to Möller. However, we are

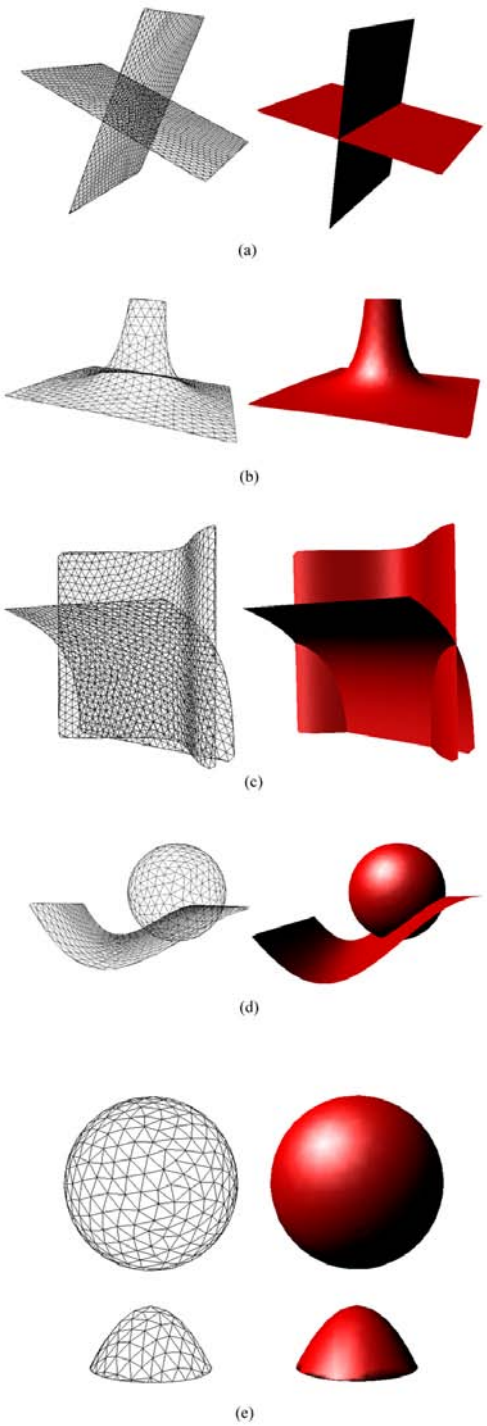

Figure 9: Examples of implicit surfaces.

not able yet to process self-intersections belonging to the same symbolic component such as, for example, the self-intersection along the positive z-axis of the Cartan-Whitney umbrella $x^2 - zy^2 = 0$. This is an open problem that can be solved using the Implicit Function Theorem and symbolic processing to compute such 1-dimensional singularities. The same also applies to 0-dimensional singularities such as *isolated singularities* (e.g. the apex of the double cone) and *isolated points*.

In fact, as a consequence of the Implicit Function Theorem, we can say that the singularities or singular points of a surface $f(x,y,z) = 0$ are those at which all the partial derivatives simultaneously vanish. For example, the double cone $x^2 + y^2 - z^2 = 0$ consists of two cones placed apex to apex. Its double apex can be determined by solving the following system of equations:

$$\begin{cases} \frac{\partial f}{\partial x} = 0 \\ \frac{\partial f}{\partial y} = 0 \\ \frac{\partial f}{\partial z} = 0 \end{cases}$$

The apex of the double cone is then the origin $(0,0,0)$. Found this 0-dimensional singularity, we can devise a strategy around it to locate two seeding points, one on each single cone. The first is determined by using the Newton corrector from an arbitrary point inside the bounding box. The second requires a starting point near the second cone in order to guarantee it converges to a point of the second cone.

Looking at other continuation methods it seems that their authors were not concerned about these issues, namely surfaces with various components, singularities, and sign-invariant components. Their focus seems to be on the quality of the mesh, i.e. aesthetics aspects of the mesh.

Recall that, according to Bloomenthal [Bloomenthal 1997], there are two major classes of continuation methods: *piecewise linear continuation methods* and *predictor-corrector continuation methods*. In the first class, the mesh growth results from the expansion of cells (typically cubes or tetrahedra) straddling the surface, being the surface sampled by a 2-point numerical method that outputs the intersection points between the cell edges and the surface itself [Allgower and Schmidt 1985] [Allgower and Gnutzmann 1987] [Dobkin et al. 1990].

In the second class, the mesh grows by creating new vertices and their triangles beyond the current polygonization's border. The initial predicted position of each vertex is on the tangent plane at the border, which is then settled onto the implicit surface, its corrected position. The correction is done by some numerical method. We use the Newton-Raphson corrector that is a 1-point numerical method, but we could have used a 2-points corrector similar to that employed by Karkanis and Stewart [Karkanis and Stewart 2001]. New polygons are then attached to the current polygonization border to join the vertex [Rheinboldt 1988]. Alternatively, a tangent plane's disk centered on a border vertex may be created with an inscribed hexagon of triangles, projected onto the surface, and merged with the current mesh [Henderson 1993] [Henderson 2002]. As described above, we basically use the Henderson's disk with some additional heuristics to obtain an approximately uniform partition of the external angle and, consequently, a mesh with regular triangles (i.e. approximately equilateral triangles). Hartmann [Hartmann 1998] and Karkanis and Stewart [Karkanis and Stewart 2001] use similar heuristics, although Karkanis and Stewart also uses the curvature criterion to produce good triangles in those surface regions where the triangle size changes quickly.

8 Conclusions and Future Work

The first goal of our work was to design a comprehensive algorithm for implicit surfaces, and implement it as quickly as possible using Java technology. We were not concerned about performance issues such as rendering speed and time.

The second goal of our work was to design and implement an algorithm capable of rendering a wide range of implicit surfaces, in particular those having sign-invariant components that are undetected by 2-points numerical methods during the sampling stage.

The third goal was to render surfaces having two or more components, independently of whether they were connected or not. In this respect, decomposing a surface into components by factorizing its function expression is an important tool. As a consequence, many non-manifold surfaces can be sampled and rendered.

In the near future, we hope to overcome the problem of self-intersections of surface components (e.g. Whitney umbrella is an irreducible surface that crosses itself along the positive z-axis), as well as to resolve other 0- and 1-dimensional singularities such as cusps and dangling line segments.

Acknowledgements

We are very grateful to Francisco Morgado for his valuable criticism during the development of the algorithm.

This research work was sponsored by the Fundação para a Ciência e Tecnologia, under the Project No. POSC/EIA/63046/2004.

References

ALLGOWER, E., AND GNUTZMANN, S. 1987. An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces. *SIAM Journal on Numerical Analysis 24*, 2, 452–469.

ALLGOWER, E., AND SCHMIDT, P. 1985. An algorithm for piecewise linear approximation of an implicitly defined manifold. *SIAM Journal on Numerical Analysis 22*, 2, 322–346.

BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Computer Aided Geometric Design 4*, 5, 341–355.

BLOOMENTHAL, J. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc.

BRODZIK, M. 1998. he computation of simplicial approximations of implicitly defined p-manifolds. *Computers and Mathematics with Applications 36*, 6, 389–423.

DOBKIN, D., LEVY, S., THURSTON, W., AND WILKS, A. 1990. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics 9*, 4, 389–423.

GONZALEZ-VEGA, L., AND NECULA, I. 2002. Efficient topology determination of implicitly defined algebraic plane curves. *Computer Aided Geometric Design 19*, 9 (December), 719–743.

HALL, M., AND WARREN, J. 1990. Adaptive polygonization of implicitly defined surfaces. *IEEE Computer Graphics and Applications 10*, 6, 33–42.

HARTMANN, E. 1998. A marching method for the triangulation of surfaces. *The Visual Computer 14*, 3, 95–108.

HENDERSON, M. 1993. Computing implicitly defined surfaces: Two parameter continuation. IBM Research Research Report RC18777, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, USA.

HENDERSON, M. 2002. Multiple parameter continuation: computing implicitly defined k-manifolds. *International Journal of Bifurcation and Chaos 12*, 3, 451–476.

KARKANIS, T., AND STEWART, A. 2001. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications 22*, 2 (March/April), 60–69.

LI, Q., WILLS, D., PHILLIPS, R., VIANT, W., GRIFFITHS, J., AND WARD, J. 2004. Implicit fitting using radial basis functions with ellipsoid constraint. *Computer Graphics Forum 23*, 1.

LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3-d surface construction algorithms. *Computer Graphics 21*, 4, 163–169.

MELVILLE, R., AND MACKEY, D. 1995. New algorithm for two-dimensional numerical continuation. *Computers and Mathematics with Applications 30*, 1, 31–46.

MÖLLER, T. 1997. A fast triangle-triangle intersection test. *Journal of Graphics Tools 2*, 2, 25–30.

MOURRAIN, B., AND TÉCOURT, J.-P. 2005. Computing the topology of real algebraic surface. In *MEGA Electronic Proceedings*.

MULLER, H., AND STARK, M. 1993. Adaptive generation of surfaces in volume data. *The Visual Computer 4*, 9, 182–199.

RHEINBOLDT, W., AND BURKARDT, J. 1983. A locally parameterized continuation process. *ACM Transactions on Mathematical Software 9*, 236–246.

RHEINBOLDT, W. 1987. On a moving frame algorithm and the triangulation of equilibirum manifolds. In *ISNM79: Bifurcation: Analysis, Algorithms, Applications*, T. Kuper, R. Seydel, and H. Troger, Eds., 256–267.

RHEINBOLDT, W. 1988. On the computation of multi-dimensional solution manifolds of parameterized equations. *Numerische Mathematik 53*, 165–182.

SCHMITT, F., BARSKY, B., AND DU, W. 1986. An adaptive subdivision method for surface-fitting from sampled data. *Computer Graphics 20*, 4 (July), 179–188.

SEIDEL, R., AND WOLPERT, N. 2005. On the exact computation of the topology of real algebraic curves. In *Proceedings of the 21st ACM Symposium on Computational Geometry*, ACM Press, 107–115.

STANDER, B., AND HART, J. 1997. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Computer Graphics 31*, 3 (August), 279–286.

VELHO, L., FIGUEIREDO, L., AND GOMES, J. 1999. A unified approach for hierarchical adaptive tesselation of surfaces. *ACM Transactions on Graphics 18*, 4, 329–360.

ZHOU, J., N.PATRIKALAKIS, TUOHY, S., AND YE, X. 1997. Scattered data fitting with simplex splines in two and three dimensional spaces. *The Visual Computer 13*, 7, 295–315.