University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitní 8
306 14 Plzeň
Czech Republic

Methods for Implicit Surfaces Polygonization

State of the Art and Concept of Doctoral Thesis

Martin Čermák

Technical Report No. DCSE/TR-2003-01
January, 2003

Distribution: public

Technical Report No. DCSE/TR-2003-01
January, 2003

Methods for Implicit Surfaces Polygonization

Martin Čermák

Abstract

Both object modeling and visualization belong to the fundamental tasks of the computer graphics. In recent years, implicit modeling has become attractive. Because of the fact that the implicit surfaces conveniently define volumes, they are frequently used in CSG-based solid modelers. The visualization of objects defined in such way is possible either by direct rendering based on Ray-tracing principle or by approximation of the implicit models by polygons, triangular mesh usually. Such approximation process is called polygonization. The polygonal (triangular) meshes are supported by a wide range of graphics hardware and, therefore, working with them is very fast as well as their arbitrarily transformations are possible without repeated solution of the implicit function. Programs for 3D graphics support polygonal meshes as well. It is not complicated to import such object and also its additional modification is possible with these professional tools.

The offered work contains an overview of commonly used principles for polygonization of implicit objects as well as information about our previous research in this field. Advantages and disadvantages of presented algorithms are discussed. The outlook of our future work is presented in conclusion.

This work was supported by the Ministry of Education of the Czech Republic – project MSM 235200005.

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
http://herakles.zcu.cz/publications.php
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitní 8
306 14 Plzeň
Czech Republic

Abstract

Both object modeling and visualization belong to the fundamental tasks of the computer graphics. In recent years, implicit modeling has become attractive. Because of the fact that the implicit surfaces conveniently define volumes, they are frequently used in CSG-based solid modelers. The visualization of objects defined in such way is possible either by direct rendering based on Ray-tracing principle or by approximation of the implicit models by polygons, triangular mesh usually. Such approximation process is called polygonization. The polygonal (triangular) meshes are supported by a wide range of graphics hardware and, therefore, working with them is very fast as well as their arbitrarily transformations are possible without repeated solution of the implicit function. Programs for 3D graphics support polygonal meshes as well. It is not complicated to import such object and also its additional modification is possible with these professional tools.

The offered work contains an overview of commonly used principles for polygonization of implicit objects as well as information about our previous research in this field. Advantages and disadvantages of presented algorithms are discussed. The outlook of our future work is presented in conclusion.

Contents

Chapter 1

Introduction

The use of real functions of several variables for defining geometric objects is quite common in mathematic and computer science. Functionally represented volumes and surfaces appear to be useful in solid modeling, computer aided geometric design (CAGD), animation, range data processing and volume graphics.

Implicit surfaces are two-dimensional, geometric shapes that exist in three-dimensional space. An implicit surface is mathematically defined by the equation $f(\mathbf{p}) = 0$, where $\mathbf{p} = [p_x, p_y, p_z]$ is a point in three-dimensional Euclidean space. An iso-surface is a similar set of points for which $f(\mathbf{p}) = c$, where *c* is the iso-contour value of the surface. More precise mathematical definition is described in [37]. A subset $O \subset \Re^n$ is called an *implicit object* if there exists a function $f : U \to \Re^k$, $O \subset U$, and a subset $V \subset \Re^k$, such that $O = f^{-1}(V)$. That is:

$$O = \{ \mathbf{p} \in U : f(\mathbf{p}) \in V \}. \tag{1}$$

There are two different definitions for implicit objects. The first one [3], [4], [5] defines an implicit object as $f(\mathbf{p}) < 0$ and the second one, F-rep (functional representation) [17], [28], [35], defines it as $f(\mathbf{p}) \geq 0$. These inequalities describe a half space in E^3. An object defined by these inequalities is usually called solid (or volume).

If *f* is an arbitrary procedural method (i.e. a 'black-box' function that evaluates **p**) then the geometric properties of the surface can be deduced only through numerical evaluation of the function.

The implicitly defined object can be bounded (finite in size), such as a sphere, or unbounded, such as a plane. The value of *f* is often a measure of distance between **p** and the surface. The measure is Euclidean if it is ordinary (physical) distance. For an algebraic surface, *f* measures algebraic distance.

Because an implicit representation does not produce points by substitution, root-finding has to be employed to render its surface. One such method is ray-tracing [12], which generates excellent photo-realistic images of implicit objects. Alternatively, an image of the function can be created with volume rendering.

Relation to parametric surfaces

Both parametric and implicit methods are well developed in computer graphics. Traditionally, computer graphics has favored polynomial parametric over implicit surfaces because they are simpler to render and more convenient for geometric

operations such as computing curvature and controlling position and tangency. Parametric surfaces are generally easier to draw, tessellate, subdivide, bound, and navigate along.

An implicit surface naturally describes an object's interior, whereas a comparable parametric description is usually piecewise. The ability to enclose volume and to represent blends of volumes provides a straightforward (although less precise) implicit alternative to fillets, rounds, and other 'free-form' parametric surfaces that require care in joining so that geometric continuity is established along the seams. Consequently, animations of organic shapes commonly employ implicit surfaces.

Point classification (determining whether a point is inside, outside, or on a surface) is simpler with implicit surfaces, depending only on the sign of *f*. This facilitates the construction of complex objects from primitive ones and simplifies collision detection.

Certain shapes may be described exactly in both parametric and implicit form, as demonstrated for the unit circle, [6]. The three-dimensional case is:

$$\begin{array}{ll} \text{trigonometric} & x = (\cos(\alpha)\cos(\beta),\ y = \sin(\alpha),\ z = \cos(\alpha)\sin(\beta),\ \alpha\in[0,\pi],\ \beta\in[0,2\pi) \\ \text{rational} & x = 4st/w,\ y = 2t(1-s^2)/w,\ z = (1-t^2)(1+s^2)/w, \\ & \text{for } w = (1+s^2)(1+t^2),\ s,t\in[0,\ 1] \\ \text{implicit} & f(x,y,z) = x^2+y^2+z^2-1 \end{array} \qquad (2)$$

Points on the parametrically defined sphere are readily found by substitution of α and β into the equations for *x*, *y*, *z* (similarly for *s* and *t*). By sweeping (α,β) through its domain in E^2, points along the entire surface are conveniently generated for display, piecewise approximation, etc. This natural conversion from the parametric (two-dimensional) space of a surface to the geometric (three-dimensional) space of an object is a fundamental convenience. There is no comparable mechanism for implicit surfaces (unless the implicit equation is reduced to two explicit equations, as is possible for some low degree algebraic surfaces).

The surface normal for a regular point on an implicit surface is computed as the unit-length gradient; the normal to a parametric surface is usually computed as the cross-product of the surface tangents in the two parametric directions.

The class of algebraic surfaces subsumes that of rational parametric surfaces. Thus, implicit surfaces are more likely to be closed under certain operations than their parametric counterparts. For example, the offset surface from an implicit surface remains an implicit surface, whereas the offset from a parametric surface is, in general, not parametric. Because parametric and implicit forms have complementary advantages, it is useful to convert from one form to the other.

Conversion from parametric to the implicit form is known as *implicitization*, and may be performed on any rational parametric surface (or curve). This is accomplished by elimination of the parameters in the parametric form. For example, elimination of *s* and *t* from the rational equations yields the implicit form in *x*, *y*, *z*.

The conversion from implicit to parametric form is known as *parameterization*. Associating a point (x,y,z) with its equivalent parametric position (s,t) is known as *inversion*. Parameterization is not always possible because implicit surfaces defined by certain polynomials of fourth and higher degree cannot be parameterized by rational

functions. Conversion is always possible for non-degenerate quadrics and for cubics that have a singular point, [6].

Continuity, Differentiability and Manifoldness

In order that normals are defined along an implicit surface, the function *f* must be continuous and differentiable. That is, the first partial derivatives $F_x = \partial f/\partial x$, $F_y = \partial f/\partial y$, $F_z = \partial f/\partial z$ must be continuous and not all zero, everywhere on the surface. Such a function is known as analytic (or is considered analytic in a region that is differentiable). When given as an ordered triplet, the partials define the gradient ∇f of the function. The unit-length gradient is usually taken as the surface normal.

$$\mathbf{n} = (n_x, n_y, n_z) = \left(\frac{F_x}{J}, \frac{F_y}{J}, \frac{F_z}{J} \right), \text{ where } J = \sqrt{F_x^2 + F_y^2 + F_z^2} \ . \tag{3}$$

For a 'black-box' or other non-differentiable function, the gradient may be approximated numerically using forward differences and some discrete step size Δ:

$$\nabla f(\mathbf{p}) \cong \big(f(\mathbf{p} + \Delta x) - f(\mathbf{p}), f(\mathbf{p} + \Delta y) - f(\mathbf{p}), f(\mathbf{p} + \Delta z) - f(\mathbf{p})\big) / \Delta, \tag{4}$$

where Δx, Δy, and Δz are displacements by Δ along the respective axes. For small Δ, the error is proportional to Δ. If ∇f is computed by central differences:

$$\nabla f(\mathbf{p}) \cong \big(f(\mathbf{p} + \Delta x) - f(\mathbf{p} - \Delta x), f(\mathbf{p} + \Delta y) - f(\mathbf{p} - \Delta y), f(\mathbf{p} + \Delta z) - f(\mathbf{p} - \Delta z)\big) / 2\Delta, \tag{5}$$

the error is proportional to Δ^2, [6].

If the gradient is non-null at a point **p**, then **p** is said to be *regular* (or simple) and $\nabla f(\mathbf{p})$ is normal (perpendicular) to the surface at **p**. If, however, the gradient (or, equivalently, the tangent vector) is indeterminate, the point is *singular* (also called critical or non-regular), [26]. For example, the cone $f(x,y,z) = -x^2+y^2+z^2$ is regular with the exception of a singularity at the origin **S**, see Figure 1. The normal at a singular point is sometimes given as the average of the normals of surrounding vertices.

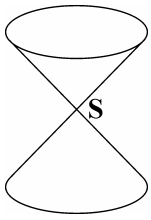

Figure 1. The apex of a cone is a singular point.

If the surface is regular and the second partial derivatives are continuous, then the surface has continuous curvature (the surface is G^2 continuous). Furthermore, if the surface is regular, it defines a topological manifold and such implicit object is also called *valid*, [37].

The 2-manifold is a fundamental concept from algebraic and differential topology. It is a surface embedded in E^3 such that the infinitesimal neighborhood around any point on the surface is topologically equivalent ('locally diffeomorphic') to a disk. Intuitively, the surface is 'watertight' and contains no holes or dangling edges. Typically, the manifold is bounded (or closed). For example, a plane is a manifold but is unbounded and thus not watertight in any physical sense. A manifold-with-boundary is a surface locally approximated by either a disk or a half-disk. All other surfaces are non-manifold, see Figure 2.

Figure 2. Manifold, manifold with boundary, and non-manifold surface, the picture is taken from [6].

From the implicit function theorem it may be shown that for $f(\mathbf{p}) = 0$, where *0* is a regular value of *f* and *f* is continuous, the implicit surface is a two-dimensional manifold. The Jordan-Brouwer Separation Theorem states that such a manifold separates space into the surface itself and two connected open sets: an infinite `outside' and a finite 'inside', [6].

Consider two examples for which no manifold exists. The first is simply $f(\mathbf{p}) = 0$. Here, ∇f is everywhere *0*, there is no `inside' nor `outside' and no boundary between the two. The second is a degenerate sphere $f(x,y,z) = x^2+y^2+z^2$. Here, $\nabla f = (2x,2y,2z)$, which is null at the origin, the only point satisfying *f*. Intuitively, the `inside' is degenerate. Whether or not a surface is manifold concerns its polygonal representation.

Surface curvature

The Hessian

The Hessian form associated with a function $f(x_1,x_2,...,x_n)$ is the matrix of second-order partial derivatives of *f* with respect to x_i:

$$Hf(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}. \tag{6}$$

The Hessian indicates the rate of change in the gradient of *f* and will be useful for, among other things, computing the curvature of implicit objects, [37].

The Gauss map

We know that for curves the curvature at a point **p** is measured by a number. For surfaces, it is measured by a map.

Let *M* be an oriented codimension-1 sub-manifold in $\Re^{n+1}$. Denote by N(**p**) the unit normal vector to *M* at **p**. The Gauss map, $N : M \to S^n$, associates to each $\mathbf{p} \in M$ the point N(**p**) on the unit n-dimensional sphere S^n, see Figure 3.

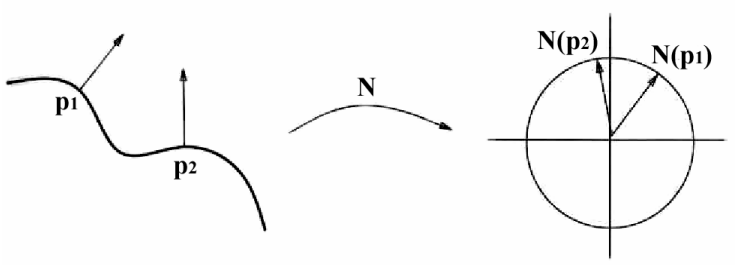

Figure 3. The Gauss map, taken from [37].

The derivative N' of N is a measure of how the normal vector is changing. Because N is a unit vector, N' indicates the change in its direction, and therefore N' conveys information about the curvature of the surface, [37]. It is easy to show that:

- $N'(\mathbf{p})$ is a linear operator on T_pM,
- $N'(\mathbf{p})$ is self-adjoint.

$N'(\mathbf{p})$ is sometimes called the *Weingarten map* in the literature.

The fundamental forms

For any self-adjoint linear transformation on a vector space with dot product there is a real-valued function $O(\mathbf{v}) = N(\mathbf{v}) \cdot \mathbf{v}$ called the Quadratic form associated with *N*.

The *First fundamental form* of *M* at **p** is the quadratic form F_p associated with the identity transformation on T_pM.

$$F_p(\mathbf{v}) = \mathbf{v} \cdot \mathbf{v} \tag{7}$$

Therefore, this quadratic form defines the inner product in each tangent plane to the surface. All the metric properties of the surface are connected to it.

The *Second fundamental form* of *M* is the quadratic form S_p associated with the Weingarten map N_p at a point **p**.

$$S_p(\mathbf{v}) = N'_p(\mathbf{v}) \cdot \mathbf{v} \tag{8}$$

A surface is completely determined up to rigid motion by its first and second fundamentals forms, [26], [31].

If $M = f^{-1}(c)$ is a regular implicit surface in $\Re^{n+1}$ with orientation given by the normal vector field and $\mathbf{v} = (v_1,...,v_{n+1})$ is a tangent vector to *M* at a point $\mathbf{p}$, $\mathbf{v} \in T_pM$, the second fundamental form is related to the Hessian form of *f*. More precisely in matrix notation:

$$S_p(\mathbf{v}) = \frac{1}{\|\nabla f(\mathbf{p})\|} \mathbf{v}^T \cdot Hf(\mathbf{p}) \cdot \mathbf{v} \tag{9}$$

Surface curvature

The second fundamental form allows us to investigate the curvature of a surface. The *Normal curvature* of *M* at $\mathbf{p}$ in the direction $\mathbf{v}$ is defined by

$$k(\mathbf{v}) = S_p(\mathbf{v}) = \langle N_p'(\mathbf{v})\mathbf{v} \rangle, \text{ when } \|\mathbf{v}\| = 1. \tag{10}$$

In other words, $k(\mathbf{v})$ is equal to the normal component of acceleration of any curve, contained in *M*, passing through $\mathbf{p}$ with velocity $\mathbf{v}$.

Because $N'(\mathbf{p})$ is a self-adjoint linear transformation of T_pM, there exists an orthonormal basis $\mathbf{v}_1,...,\mathbf{v}_n$ of T_pM whose vectors, $\mathbf{v}_i$, are eigenvectors of $N'(\mathbf{p})$. The eigenvalues $k_1(\mathbf{p}),..., k_n(\mathbf{p})$ of $N'(\mathbf{p})$ are called *principal curvatures* of *M* at $\mathbf{p}$ and the correspondent unit eigenvectors of $N'(\mathbf{p})$ are called *principal directions*. The principal curvatures are stationary values of normal curvature $k(\mathbf{p})$ and among them $k(\mathbf{p})$ attains its minimum and maximum values.

In general, we can diagonalize the Hessian matrix *H* to obtain the eigenvalues and eigenvectors of $N'(\mathbf{p})$. Alternatively, the following formulas allow us to compute the principle curvatures k_i and the principle directions $\mathbf{v}_i$ directly from *H*, [37].

$$k_i = \frac{\mathbf{a}^T H\mathbf{a} + \mathbf{b}^T H\mathbf{b} \pm \sqrt{\left(\mathbf{a}^T H\mathbf{a} - \mathbf{b}^T H\mathbf{b}\right)^2 + 4\left(\mathbf{a}^T H\mathbf{b}\right)^2}}{2\|\nabla f\|}, \tag{11}$$

$$\mathbf{v}_i = \begin{pmatrix} a_1 + b_1 \dfrac{\|\nabla f\| k_i - \mathbf{a}^T H\mathbf{a}}{\mathbf{b}^T H\mathbf{a}} \\ a_2 + b_2 \dfrac{\|\nabla f\| k_i - \mathbf{a}^T H\mathbf{a}}{\mathbf{b}^T H\mathbf{a}} \\ a_3 + b_3 \dfrac{\|\nabla f\| k_i - \mathbf{a}^T H\mathbf{a}}{\mathbf{b}^T H\mathbf{a}} \end{pmatrix}, \tag{12}$$

for i = 1, 2, where

$$\mathbf{a} = \left(\frac{1}{\gamma} \frac{\partial f}{\partial x_2}, -\frac{1}{\gamma} \frac{\partial f}{\partial x_1}, 0 \right)^T, \quad \mathbf{b} = \left(\frac{1}{\gamma \|\nabla f\|} \frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_3}, \frac{1}{\gamma \|\nabla f\|} \frac{\partial f}{\partial x_2} \frac{\partial f}{\partial x_3}, \frac{-\gamma}{\|\nabla f\|} \right)^T. \tag{13}$$

The trace and determinant of the Gauss map are important intrinsic properties of a surface.

The *mean curvature* $\overline{K}(\mathbf{p})$ of M at $\mathbf{p}$ is 1/n times the trace of $S(\mathbf{p})$:

$$\overline{K}(\mathbf{p}) = \text{trace}\, S(\mathbf{p}) = \frac{1}{n}\sum_{i}^{n} k_i(\mathbf{p}) \tag{14}$$

It is the average value of the principal curvatures at $\mathbf{p}$.

The determinant of $S(\mathbf{p})$ is called the *Gauss-Kronecker curvature* K_G of M at $\mathbf{p}$.

$$K_G(\mathbf{p}) = \det S(\mathbf{p}) = \prod_{i}^{n} k_i(\mathbf{p}) \tag{15}$$

It is equal to the product of the principal curvatures.

Chapter 2

State of the Art

Modeling of Implicit objects

Constructive Solid Geometry

With Constructive Solid Geometry (CSG), an object is evaluated 'bottom-up' according to a binary tree. The leaf nodes are usually restricted to low degree polynomial primitives, such as spheres, cylinders, ellipsoids, half-spaces, and tori. The internal nodes represent Boolean set operations.

The primitives in CSG may be represented implicitly and combined by set-theoretic Boolean operations, [40]. These operations may create hard-edged functions that conventional polygonizers cannot accurately approximate, see Figure 4.

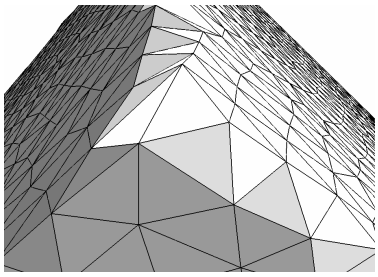

Figure 4. A corner of a cube modeled as intersection of six half-spaces.

The exact analytical definitions of the set-theoretic operations of functionally described objects have been proposed in the theory of R-functions, [28], and applied for solving problems of mathematical physics.

Let the geometric object G_1 be defined as $f_1(x,y,z) \geq 0$ and the geometric object G_2 be defined as $f_2(x,y,z) \geq 0$. The resultant object will have the defining function as follows:

R-union	$f_3 = f_1 \mid f_2$
R-intersection	$f_3 = f_1 \,\&\, f_2$
R-subtraction	$f_3 = f_1 \setminus f_2$

One of the possible analytical descriptions of R-functions is:

$$f_1 \mid f_2 = \frac{1}{1+\alpha}\left(f_1 + f_2 + \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2}\right),$$
$$f_1 \,\&\, f_2 = \frac{1}{1+\alpha}\left(f_1 + f_2 - \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2}\right), \qquad (16)$$

where $\alpha=\alpha(f_1,f_2)$ is an arbitrary continuous function satisfying the conditions $-1 < \alpha(f_1,f_2) \le 1$, $\alpha(f_1,f_2) = \alpha(f_2,f_1) = \alpha(-f_1,f_2) = \alpha(f_1,-f_2)$.

The expression for the subtraction operation is $f_1 \setminus f_2 = f_1 \& (-f_2)$. Note that with this definition of the subtraction, the resultant object includes its boundary. If $\alpha=1$, the functions (16) become:

$$f_1 \mid f_2 = \min(f_1,f_2)$$
$$f_1 \,\&\, f_2 = \max(f_1,f_2) \qquad (17)$$

This is the particular case, the functions are very convenient for calculations but have C^1 discontinuity when $f_1 = f_2$. If $\alpha=0$, the functions (16) take the most useful in practice form:

$$f_1 \mid f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2}$$
$$f_1 \,\&\, f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2} \qquad (18)$$

The functions above have C^1 discontinuity only in points where both arguments are equal to zero. If C^m continuity is to be provided, one may use another set of R-functions:

$$f_1 \mid f_2 = \left(f_1 + f_2 + \sqrt{f_1^2 + f_2^2}\right)\left(f_1^2 + f_2^2\right)^{\frac{m}{2}}$$
$$f_1 \,\&\, f_2 = \left(f_1 + f_2 - \sqrt{f_1^2 + f_2^2}\right)\left(f_1^2 + f_2^2\right)^{\frac{m}{2}} \qquad (19)$$

The more examples of set-theoretic operations, such as blending (linear, hyperbolic, super-elliptic), offsetting, bijective mapping, affine mapping, projection, Cartesian product and metamorphosis can be found in [9], [11], [24], [25], [38].

Skeleton based modeling

The skeleton is a collection of elements, each of which generates a volume. Within an implicit context, such a volume is called a skeletal primitive, which is denoted by $f_i(\mathbf{p})$, for skeletal element *i*. Thus, *f* is a function from E^3 (or E^2 for illustrative purposes) to E^1, and, usually, is C^1 continuous. The implicit surface function may be a blend of these primitives, i.e., $f(\mathbf{p}) = g(\mathbf{p}, f_1,f_2,...,f_n) = 0$, and the implicit surface is the covering, or manifold, of the skeleton, [4].

When used in a biological context, ‘skeleton’ usually refers to the rigid, mechanical support system found in most animals. In such a system, a subordinate element rotates with respect to a superior one.

Although an organism’s inner structure need not be organized hierarchically, for our purposes we assume that a skeleton is topologically equivalent to a directed acyclic graph. Such a graph, or tree, organizes the internal components of an object and is, therefore, a powerful means for the representation and manipulation of the object. The basic data structure for a skeleton, which we call an element (or, sometimes, limb), is recursive and contains the following fields, [4]:

- parent: pointer to element
- children: list of pointer to element
- transformation from parent: matrix
- geometry: geometric object
- ancillary data: . . .

The transformation is Euclidean, allowing rotation and translation. Usually the geometry is a tapered cylinder defined by two three-dimensional endpoints and their associated radii.

Each skeletal element can readily define a surrounding volume, or primitive. Although the collection of these volumes may yield a topologically complex surface, the skeletal elements remain easily defined, articulated, and displayed.

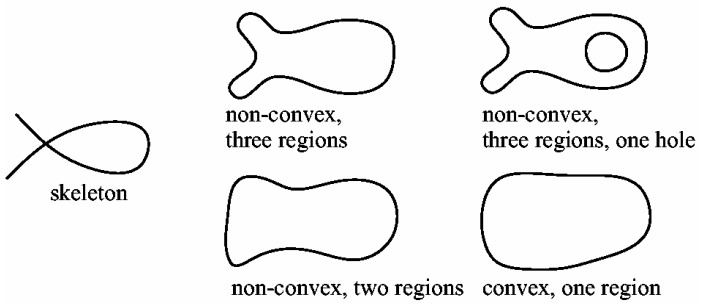

Figure 5. A skeleton and possible resulting surfaces, taken from [4]**.**

A skeleton is related to its resulting shape but its geometric complexity is not necessarily comparable to that of the shape. For example, in Figure 5, the skeleton contains a single loop. Depending on the radii associated with the skeletal elements, the resulting surface can contain a hole or not, can be convex or not, and can consist of one, two, or three convex regions.

The skeleton modeling is important for interactive modeling, [39], when a designer creates a shape by interactively defining the skeleton and various parameters that control how the skeleton becomes a polygonized surface.

Polygonal representation

For many applications it is useful to approximate an implicit surface with a mesh of triangles or polygons (formally, a discrete set of piecewise-linear, semi-disjoint elements). Conversion of a functionally specified implicit surface to a polygonal approximation can require considerable computation, but is required only once per surface and allows rendering of the surface by conventional polygon scan conversion. For differentiable *f*, [29], this is always possible because all manifold surfaces may be triangulated. Such mesh conversion is popularly known as *polygonization*.

Approximation error

The approximation error is a measure of difference between the polygonal model and its mathematical description. There are several possibilities how to evaluate this difference.

One way is to determine the distance of polygonal mesh's elements from the real (mathematically defined) surface. Let the distance between a point **x** and an implicit surface be defined as follows:

$$\mathrm{dist}(\mathbf{x}) = \min\left\{ \left\| \mathbf{x} - \mathbf{x}_z \right\| : \mathrm{f}(\mathbf{x}_z) = 0 \right\} \tag{20}$$

Then, the average error in the vertices positioning (actually, it is the error of a root finding algorithm) can be evaluated as:

$$\mathrm{E}_{av} = \frac{\sum_{i=1}^{N} \mathrm{dist}(\mathbf{v}_i)}{N}, \tag{21}$$

where $\mathbf{v}_i$ is a vertex in the triangulation and *N* is a number of vertices.

The average error of the approximation by triangles can be determined as:

$$\mathrm{E}_{at} = \frac{\sum_{i=1}^{M} \mathrm{dist}(\mathbf{t}_i)}{M}, \tag{22}$$

where $\mathbf{t}_i$ is the centre of gravity of a triangle and *M* is a number of triangles.

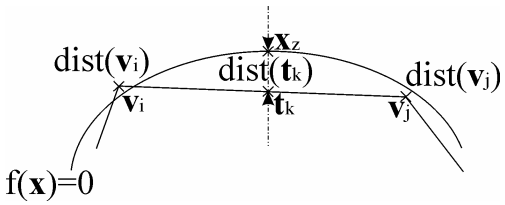

Figure 6. Approximation of an implicit object with a triangular net (contours and lines in two-dimensional example); distance of point $\mathbf{t}_k$ (the centre of gravity of a triangle) from the real surface.

Determination of the exact (real) distance of the given point to the implicit surface is computationally expensive and, therefore, several approximations are often used.

Each vertex coordinates are usually computed by an iteration process which is stopped when the function value in the given point is less then some ε. In such cases, the real distance between the surface vertex and the implicit surface is approximated by the *Algebraic distance* defined as:

$$\mathrm{dist}_A(\mathbf{x}) = \left|f(\mathbf{x})\right| \tag{23}$$

For normalized implicit functions, the Algebraic distance is equal to the real distance (Euclidian distance in Euclidian space) but in majority, it is only proportional to the real distance. For example, the Sphere implicit function is usually defined as:

$$r^2 - x^2 - y^2 - z^2 = 0\,. \tag{24}$$

The normalized version of the Sphere function is defined as:

$$r - \sqrt{x^2 + y^2 + z^2} = 0\,. \tag{25}$$

The other approximation of the real distance is the *Taubin's distance*, [33], defined as follows:

$$\mathrm{dist}_T(\mathbf{x}) = \frac{\left|f(\mathbf{x})\right|}{\left\|\nabla f(\mathbf{x})\right\|}\,. \tag{26}$$

The Taubin's distance is the first order approximation to the exact distance, but the approximate distance is also biased in some sense, [10]. If, for instance, a data point $\mathbf{x}$ is close to a critical point of the function, i.e., $\left\|\nabla f(\mathbf{x})\right\| \approx 0$, but $f(\mathbf{x}) \neq 0$, the distance becomes large which is certainly a limitation.

An alternative evaluation of the approximation error between the implicit model and its triangular mesh is the comparison of their surface areas. The usage of this measurement is limited only to implicit functions we know or we can compute their

surface area. Surface area of an implicit model approximated by triangles can be determined as:

$$S_p = \sum_{ti=1}^{M} S_{ti} \ , \tag{27}$$

where S_{ti} is the surface area of the i^{th} triangle.

Then, the relative error of the approximated model can be computed as:

$$E_s = \left| 1 - \frac{S_p}{S_m} \right|, \tag{28}$$

where S_m is the real surface area of the model defined by the implicit function. As the model is approximated by triangles we can assume that $S_p \leq S_m$.

Exhaustive enumeration

Exhaustive enumeration operates on a set of samples of *f* arranged as a regular, typically rectilinear lattice known as a scalar grid or voxel array. The samples may be experimental, such as CAT and MRI scans, or computed, as in simulations of fluid flow. The lattice is readily represented by a three-dimensional memory array, which can be filled by a hardware scanner in constant time.

Once the samples are obtained, each transverse cell is polygonized. Given c_1 and c_2, lattice neighbors of opposite sign, a surface vertex **v** is usually computed using linear interpolation:

$$\mathbf{v} = \alpha c_1 + (1-\alpha)c_2, \text{ where } \alpha = f(c_2)/\big(f(c_2) - f(c_1)\big), f(c_1), f(c_2) \neq 0 \tag{29}$$

This method is popularly known as 'marching cubes' or 'marching tetrahedra'. The standard Marching cubes (MC) and the Marching tetrahedra (MTE) algorithms [3], [4] are often used for an iso-surface extraction. These methods can be performed both the continuation schemes (see bellow) and the exhaustive enumeration approaches. The process of polygonization consists of two principal steps: partitioning the space into cells and the processing of each cell to produce polygons. Each cell is represented by a cube or by a tetrahedron. The implicit surface function is evaluated at corners.

A cell is transverse if any of its edges intersects the implicit surface (one edge endpoint evaluates negatively, the other positively). For each transverse edge, a surface vertex is computed (by the *Intermediate Value Theorem*, a point **p**: f(**p**) = 0 must exist along a transverse edge if *f* is continuous). Function *f* may be evaluated at arbitrary locations, which allows methods such as binary sectioning to compute surface vertex locations with arbitrary precision, unlike linear interpolation. These algorithms seek to minimize the number of evaluations of *f*, which may be arbitrarily demanding to evaluate.

The surface vertices belonging to the transverse edges of a cell are connected to form one or more polygons (alternatively, patches may be produced). The edges of the polygons lie within the faces of the cell. The order of vertex connectivity is often stored in a table of polarity configurations of the cell corners. For a cube (8 corners) and a tetrahedron (4 corners, i.e., a three-dimensional simplex) there are 256 and 16 possibilities, respectively. The 256 possible configurations of a cube can be reduced to only 15 fundamentals and the others can be obtained by rotation and application of symmetry. Figure 7a shows the basic 15 configurations of a cube and the configurations of a tetrahedron are shown in Figure 7b.

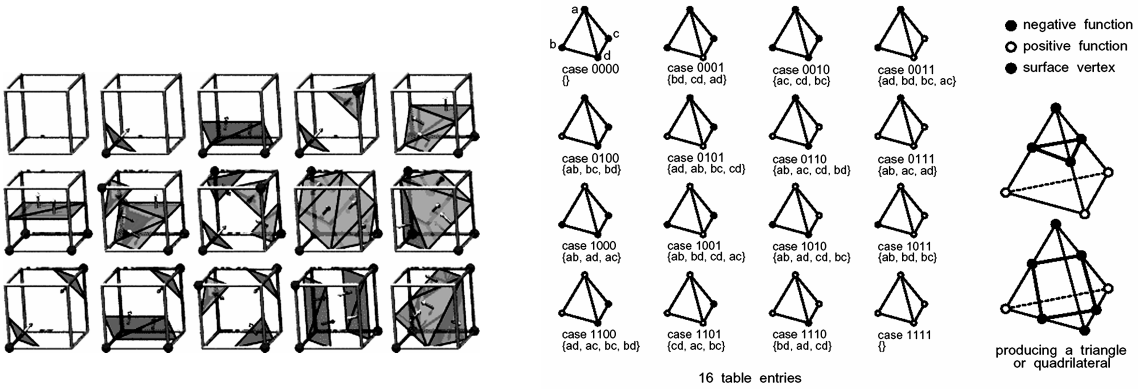

Figure 7. a) The basic 15 configuration of a cube, b) the configuration of a tetrahedron, taken from [3].

Because the tetrahedral edges include the diagonals of the cube faces, the tetrahedral decomposition yields to a greater number of surface vertices per surface area than the cubical polygonization does.

The Marching cubes and the Marching tetrahedra algorithms generate a triangular mesh which is much influenced by a regular grid. Therefore, next adjustment of the mesh is suitable.

The application of the Marching cubes algorithms includes electron motion, computational electromagnetic, polypeptide visualization, biomedical visualization, molecular modeling, etc.

Piecewise-Linear continuation

Piecewise-linear principles have been applied to implicit surfaces using a tetrahedral cell and a cubic cell, [3], [34]. Beginning with a single transverse 'seed' cell, new cells are propagated across transverse faces until the entire surface is enclosed.

Because only transverse cells are generated, piecewise-linear continuation requires $O(N^2)$ function evaluations, where N is a measure of the size of the object (thus, N^2 corresponds to the object's surface area, [3], [4]), see Figure 8. In comparison, exhaustive enumeration requires $O(N^3)$ samples. Compared with subdivision, continuation appears less prone to under-sampling.

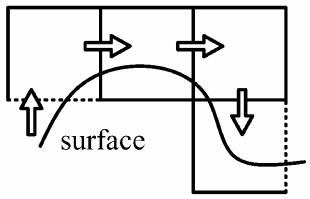

Figure 8. Continuation scheme, 2D example for illustration, taken from [3].

Exhaustive enumeration yields all disjoint surface components (with detectable size). Continuation, however, produces a single component for each seed cell; to polygonize all disjoint surface components, continuation must be performed for each, using an appropriate seed cell.

Predictor-Corrector continuation

Predictor-corrector methods [1], [14], [15], [16] apply directly to the surface, creating elements (usually triangles or polygons) by joining an initial surface point with additional points. New points are computed by displacement from a known point along the tangent plane and then corrected (e.g., using Newton iteration) onto the surface. These methods are problematic for surfaces because surface vertices are not intrinsically ordered (unlike a one-dimensional contour), which complicates detection of global overlap.

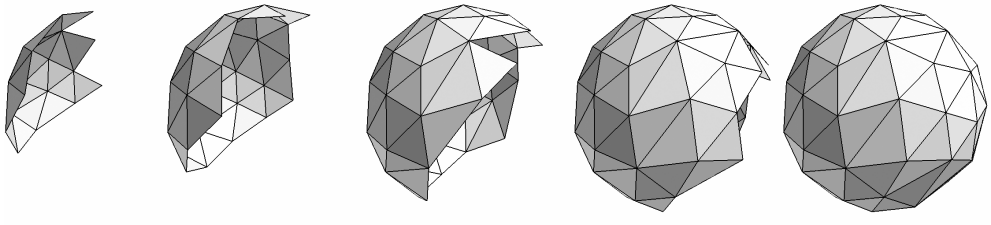

Figure 9. Continuation scheme, new triangles are directly generated on an implicit surface.

Marching triangles

The idea of the Marching triangles (MTR) algorithm, [14], consists of five steps:

Step 0: Arbitrarily choose a starting point **s** in the neighborhood of the surface and find the point $\mathbf{p}_1$ that lies on the surface. Surround $\mathbf{p}_1$ with a regular hexagon $\mathbf{q}_2,\ldots,\mathbf{q}_7$ in the tangent plane. Determine the points $\mathbf{p}_2,\ldots,\mathbf{p}_7$ corresponding to the starting points $\mathbf{q}_2,\ldots,\mathbf{q}_7$ that lie on the surface (Figure 10a). The triangles $(\mathbf{p}_1,\mathbf{p}_i,\mathbf{p}_{i+1})$ are the first six triangles of the triangulation. The ordered array of points $\mathbf{p}_2,\ldots,\mathbf{p}_7$ form the first actual front polygon[1] $\prod_0$.

[1] the border of the triangulation

Step 1: For every point of the actual front polygon $\prod_0$, determine the angle of the area till to be triangulated and form front angles (Figure 10b).

Step 2: Check if any point $\mathbf{p}_i$ of the actual front polygon is near:

a) to a point of $\prod_0$ that is different from $\mathbf{p}_i$ and its neighbors. Then divide the actual front polygon $\prod_0$ into a smaller one and an additional front polygon (Figure 11a).

b) to a point of any other front polygon $\prod_m$, m>0. Then unite the polygons $\prod_0$, $\prod_m$ to a new and larger actual front polygon (Figure 11b). Delete $\prod_m$.

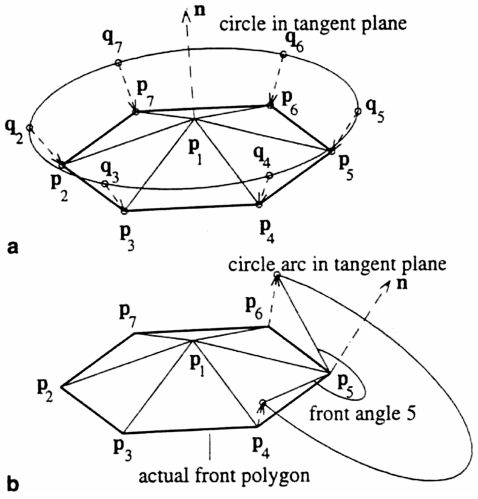

Figure 10. The first steps of the Marching triangles algorithm, taken from [14].

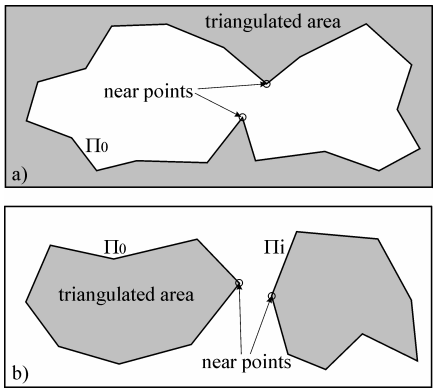

Figure 11. (a) Dividing the actual front polygon (step 2a of the MTR algorithm) and (b) uniting two front polygons (step 2b of the algorithm).

Step 3: Determine a front point $\mathbf{p}_i$ of the actual front polygon $\prod_0$ with a minimal front angle. Surround $\mathbf{p}_i$ with triangles with angles $\approx 60^0$. Delete $\mathbf{p}_i$ from the actual front polygon $\prod_0$ and insert the new points into the actual front polygon $\prod_0$.

Step 4: Repeat steps 1-3 until the actual front polygon $\prod_0$ consists of only three points that generate a new triangle. If there is another (nonempty) front polygon left, it becomes the new actual front polygon $\prod_0$ and steps 1-3 are repeated. If there are no more front polygons then the triangulation is finished.

Adaptive polygonization

Polygonization is a sampling process. If the spacing between samples is large with respect to surface curvature, detail is lost. Resolution requirements may also change with viewpoint. Any fixed sampling rate may be excessive for relatively flat regions of the surface and insufficient for relatively curved regions. If the cell size is inversely proportional to local curvature, the resulting adaptive polygonization minimizes

polygon count while maintaining geometric accuracy. Both subdivision and continuation may be performed adaptively, [1], [7], [30]. Accurate representation of non-differentiable *f*, however, may require explicit computation of its singular points.

Adaptive Marching cubes

The estimate of the surface may be improved by subdividing those cubes containing highly curved or intersecting surfaces. As is introduced in [7], using the polygon resulting from an octree node, criteria for subdivision of the node include:

- whether any edge of the cube intersects the surface,
- whether a maximum subdivision depth or a minimum cube size has been reached,
- whether more than one polygon results from the cube,
- the planarity of the polygon, and
- the divergence of vertex normals from the normal at the polygon center.

Given the polygon vertices, $\mathbf{p}_i$, their unit length normals $\mathbf{n}_i$, and the unit length normal $\mathbf{n}$ at the polygon center, the planarity of the polygon can be estimated by:

max ($\mathbf{v}_i \cdot \mathbf{n}$), $i \in [1,nPoints]$ and $\mathbf{v}_i$ the unit length vector ($\mathbf{p}_i,\mathbf{p}_{i+1}$), and the divergence of the vertex normals can be estimated by:

min ($\mathbf{n}_i \cdot \mathbf{n}$), $i \in [1,nPoints]$

Certain topological criteria, [13], warrant the subdivision of an adjacent cube. If the edge of a parent cube connects two equally signed corners and the midpoint is differently signed, as in Figure 12 left, then the three neighbors along that edge should be subdivided. For each face of a parent cube, if the four child corners that are midpoints of the four edges of the face all agree in sign but disagree with the center of the face, Figure 12 right, then the face neighbor should be subdivided. Without such subdivision, a hole will appear in the surface.

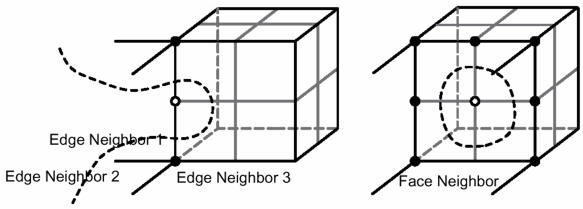

Figure 12. Conditions warranting subdivision of adjacent cubes; midpoint of an edge (left) and midpoint of a face (right), taken from [7].

The generalized cylinder in Figure 13 was created by this adaptive algorithm.

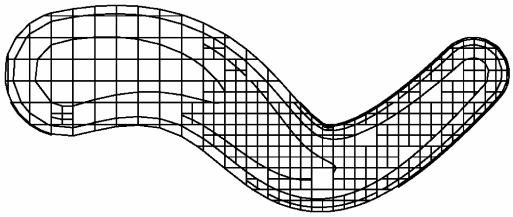

Figure 13. Adaptively subdivided generalized cylinder, taken from [7].

Adaptive Marching triangles

The algorithm introduced in [1] is based on the surface tracking approach. Starting from a seed triangle on an implicit surface, the marching triangles algorithm iteratively creates new triangles on the surface from the boundary edges. It is the improved version of the method [15] with adaptivity depending on surface curvature.

The edges of the seed triangle are inserted into the list of boundary edges. New triangles are created from the boundary edges and their new edges are appended to the end of the list, referred as L_e. Each new generated triangle has to satisfy the Delaunay property: A triangle $T(\mathbf{x}_k,\mathbf{x}_{k+1},\mathbf{x}_p)$ can be added to the mesh boundary at edge $e(\mathbf{x}_k,\mathbf{x}_{k+1})$ if no part of the surface of the existing mesh, i.e., no existing triangle, intersects the sphere centered at c_T circumscribing the triangle $T(\mathbf{x}_p,\mathbf{x}_k,\mathbf{x}_{k+1})$ with the same orientation (see Figure 14).

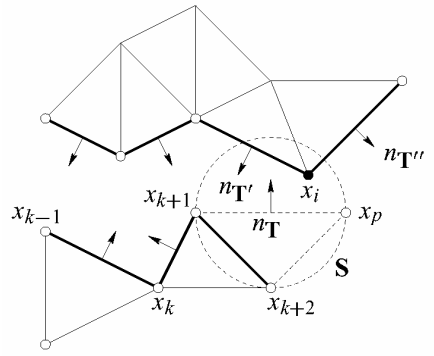

Figure 14. Creation of a new triangle T: the empty sphere criterion does not apply as the sphere *S* intersect another part of the mesh (at vertex $\mathbf{x}_i$) whose surface normals $\mathbf{n}_{T'}$ and $\mathbf{n}_{T''}$ exhibits a different orientation than $\mathbf{n}_T$. The picture is taken from [1].

The algorithm proceeds as follows, iteratively analyzing each edge $e(\mathbf{x}_k,\mathbf{x}_{k+1})$ in the list:

1. Create a new vertex $\mathbf{x}$ in the plane of the triangle $T(\mathbf{x}_i,\mathbf{x}_k,\mathbf{x}_{k+1})$ that contains the edge $e(\mathbf{x}_k,\mathbf{x}_{k+1})$. This point will be used as a first guess in the computation of the surface vertex $\mathbf{x}_p$.
2. Create a new surface vertex $\mathbf{x}_p$ by projecting $\mathbf{x}$ onto the implicit surface following the gradient of the field function ∇f .
3. Apply the Delaunay surface constraint to the new triangle $T(\mathbf{x}_p,\mathbf{x}_k,\mathbf{x}_{k+1})$ and proceed as follows:

a. If $T(\mathbf{x}_p,\mathbf{x}_k,\mathbf{x}_{k+1})$ passes the constraint, then add the triangle to the mesh and stack the edges $e(\mathbf{x}_p,\mathbf{x}_k)$ and $e(\mathbf{x}_p,\mathbf{x}_{k+1})$ to the list of edges L_e that need to be processed.
b. If $T(\mathbf{x}_p,\mathbf{x}_k,\mathbf{x}_{k+1})$ does not pass the constraint, check if one of the triangles $T(\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{x}_{k+1})$ and $T(\mathbf{x}_k,\mathbf{x}_{k+1},\mathbf{x}_{k+2})$ satisfy the Delaunay surface constraint, and modify the mesh accordingly if needed.
c. Otherwise, step over the edge $e(\mathbf{x}_k,\mathbf{x}_{k+1})$ to the next candidate edge.

4. Close the cracks that may appear in the triangulation.

The method is implemented as a single pass through the edge list L_e. Whenever the mesh growing scheme fails, the edges are left in the edge list. At the end of the algorithm, L_e forms an open contour in the polygonization. Enclosing of the left edges is in detail described in [1] and it is not necessary for understanding to the adaptive polygonization principle that is described in the next paragraphs.

In [15] the point $\mathbf{x}_p$ is computed by projecting a point $\mathbf{x}$ on the surface, where $\mathbf{x}$ is created at a constant distance d from the edge e in the plane of the triangle T. Authors in [1] used a better approach consists in adapting the parameter d to the local curvature of the surface.

Anticipating the local curvature of the field function consists of the three following steps.

1. Geometry correction step. Let $\mathbf{x}_m$ denote the mid point of the boundary edge e. At first, the point $\mathbf{x}_m$ is projected onto the implicit surface in the direction of the gradient $\nabla f(\mathbf{x}_m)$ so as to fit to the local geometry of the implicit surface, see Figure 15. This step creates a new point on the surface denoted as $\mathbf{x}_s$.

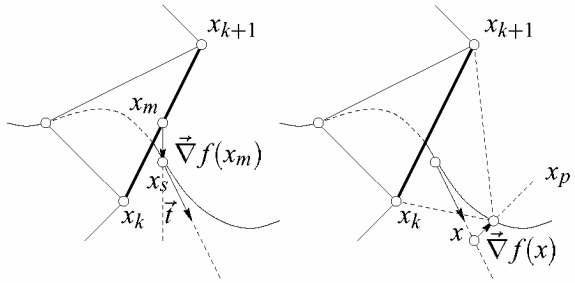

Figure 15. Characterization of the surface point $\mathbf{x}_s$ and the projected point $\mathbf{x}_p$, taken from [1].

2. Computation of the starting point. Let $\mathbf{t}$ be the unit tangent vector (see Figure 15) to the surface at the surface vertex position $\mathbf{x}_s$ defined as:

$$\mathbf{t} = \frac{\mathbf{e}_k \times \nabla f(\mathbf{x}_s)}{\left\| \mathbf{e}_k \times \nabla f(\mathbf{x}_s) \right\|}. \tag{30}$$

The point **x** may be written as $\mathbf{x}=\mathbf{x}_s+d\mathbf{t}$ where *d* is a variable distance parameter computed as:

$$d = \frac{\sqrt{3}}{2}\bar{e}, \text{ where } \bar{e} = \frac{\|\mathbf{e}_{k-1}\| + \|\mathbf{e}_k\| + \|\mathbf{e}_{k+1}\|}{3}, \tag{31}$$

and the edges $\mathbf{e}_{k-1}$, $\mathbf{e}_{k+1}$ are the neighboring edges of the edge $\mathbf{e}_k$.

The variable *d* is constrained with some limit value d_{min} and if $d<d_{min}$ then $d_{new} = 3/4d + 1/4d_{min}$.

3. Computation of the new surface vertex. Let **x** and **y** denote the two points that converge to the surface by following the gradient of the field function. The algorithm may be written as follows:

   a. Initialize **y** with the starting point **x**.
   b. While both points are on the same side of the implicit surface, i.e., f(**x**) and f(**y**) are of the same sign, perform the following sub-steps.

      - Evaluate an approximation of the distance to the surface by the Taubin's distance, equation (26).
      - Compute the new location for point **y**, marching from **x** along the direction of the gradient vector $\nabla f(\mathbf{x})$.

$$\mathbf{y} = \mathbf{x} - \alpha\frac{f(\mathbf{x})\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|^2}, \tag{32}$$

        where α is a scalar factor.

      - If f(**x**) and f(**y**) are of the same signs, store **y** in **x** and restart loop at step b.

   c. When the algorithm reaches this step, **x** and **y** are on opposite sides of the surface, so perform bisection over the line segment [**x**,**y**].

Surface refinement

One possible solution for polygonization of implicit object with sharp features is refinement of an initial triangular mesh, [2], [18], [21], [22], [36]. Simple, efficient and numerically stable algorithm is used for constructing of an initial mesh. Algorithms based on the marching cubes principle are often used. A coarsely polygonized surface is followed by subdivision of insufficiently accurate polygons. For example, if the center of a triangle is too distant from the surface, the triangle may be split at its center, which is moved to the surface. Similarly, a triangle may be divided along its edges if the divergence between surface normals at the triangle vertices is too great.

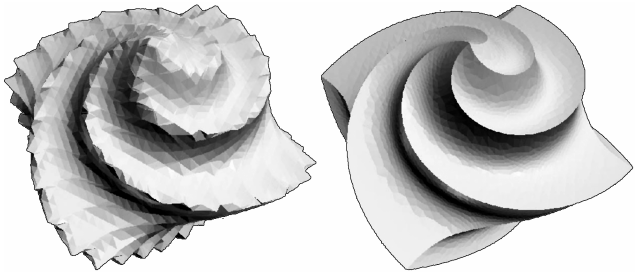

Figure 16. The initial mesh (left) created by the Marching cubes algorithm and its optimized version (right), taken from [22].

The algorithm introduced in [22] consists of following two steps. Given an implicit surface f(x,y,z) = 0 and its initial polygonization then the mesh optimization procedure is as follows.

1. Construct the dual mesh consisting of the centroid of the original mesh, modify the dual mesh by projecting its vertices onto the implicit surface, and find the tangent planes at the vertices of the modified dual mesh.
2. For each vertex, update its position by minimizing an error function equal the sum of squared distances from the vertex to tangent planes at the neighboring vertices of the modified dual mesh.

The method has several limitations. The mesh optimization process does not change the topology of an initial coarse mesh. Therefore, if fine topological details are not captured by the initial mesh, the method may produce a wrong reconstruction of the implicit surface. Another drawback of the method is a large number of calls of a function which defines the implicit surface. If the function is very complex, the method becomes computationally expensive.

Non-Manifold polygonization

Although a manifold-with-boundary may be specified by a continuous function, all points off the zero set are of the same sign. Consequently, conventional polygonization fails. A non-manifold can be implicitly represented by extending the definition of *f* to be the separation between arbitrary regions of space. A continuation method using this scheme is given in [8].

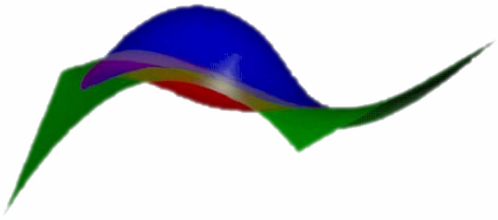

Figure 17. Polygonized non-manifold, taken from [8].

Chapter 3

Previous work and conclusions

Marching triangles improvement

Acceleration

The original algorithm described in [14] contains some parts that can be implemented more effectively. The most time-consuming part is the distances checking of the front polygons' points (Step 2 of the MTR algorithm mentioned above). Our modification of the algorithm is directed precisely towards achieving this step.

One possible solution is the subdivision of the computing area into smaller sub-areas. Each sub-area contains only one part of a set of front polygons' points. The average number of points in sub-areas depends on the sub-areas' size. Our main requirement is to minimize the number of distance checks, i.e. a selection of the most restricted set of points into which the actual front polygon can be divided or united.

The actual front polygon is divided or united only if the distance between two specified points is shorter than some limit distance σ (more information in [14]). Therefore, the most suitable choice for the size of the sub-areas side is σ, i.e. the shape of sub-areas is a cube. For this choice, the distance checks (FDC and SDC) can be accomplished only with front polygons' points which lie in adjacent sub-areas of the new point's area. Figure 18 shows a distance check for a new included point (for illustration only the E^2 example).

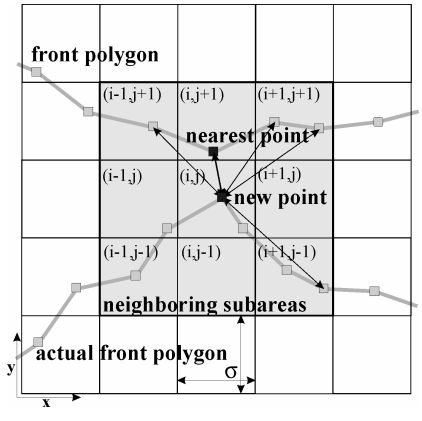

Figure 18. Space subdivision scheme.

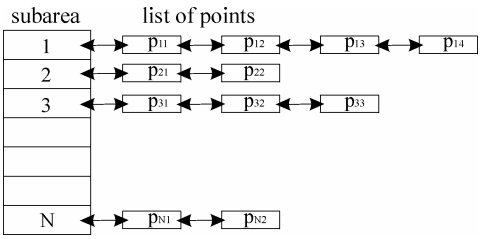

Figure 19. The data structure for the space subdivision scheme.

Each sub-area contains a list of front polygons' points located inside. The data structure used for subdivision scheme, is shown in Figure 19. Each front polygon also has its own set of points (similar as above) and each point contains one pointer to its sub-area and one pointer to its front polygon as well.

Edge detection

Detection of sharp edges is a modification of the MTR method (mentioned above) at the step of finding location of a new point (in step 3 of the MTR algorithm). The principle of the algorithm is in knowledge of normal vectors in points **p** and **q**. The point **p** already has its own accurate position and the point **q** lies in the tangent plane of the point **p**. Then the algorithm is as follows.

5. initialization, **a** = **p**, **b** = **q**, $\mathbf{n}_a$ … normal vector in the point **a**, $\mathbf{n}_b$ … normal vector in the point **b**
6. **c** = 0.5*(**a**+**b**) … binary subdivision between the points **a**, **b**
7. let the normal vector in the point **c** be $\mathbf{n}_c$, and α be the angle between vectors $\mathbf{n}_a$ and $\mathbf{n}_c$
8. if $\alpha > \alpha_{lim}$ then **b** = **c** else **a** = **c**
9. if the distance between points **a**, **b** is less then some ε, the desired point is **b**, else return to step 2

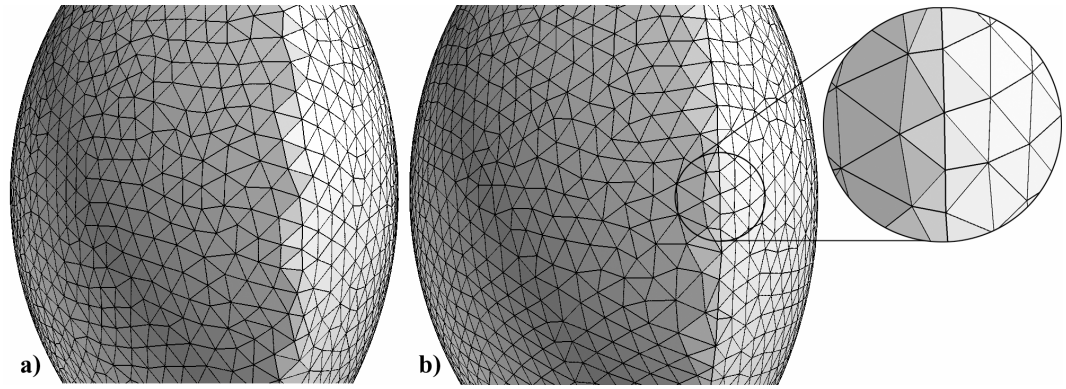

Figure 20. The implicit object modeled as intersection of two spheres; polygonized a) without edge detection; b) with edge detection algorithm.

Note: The algorithm works well, see Figure 20, but it is simple and does not contain another improving of the shape of triangles lying near the edge, see Figure 20b, the triangles are deformed by the edge.

Edge spinning algorithm and its acceleration

We have developed a new algorithm for polygonization of the implicit surfaces. The Edge spinning (ES) method put emphasis on the shape of triangles generated and on the

polygonization speed. The algorithm is a variant of marching triangles methods [1], [14], [15], [16], i.e. it is based on the continuation (surface tracking) scheme.

Data structures

The presented algorithm works only with the standard data structures used in computer graphics. The main data structure is an edge used as a basic building block for the polygonization. We use the standard winding edge and therefore, the resulting polygonal mesh is correct and complete with neighborhood among all generated triangles. If a triangle's edge lies on the triangulation border, it is contained in the list of active edges (dynamically allocated list) and it is called as an active edge. Each point contained in an active edge has two pointers to its left and right active edge (left and right directions are in active edges' orientation).

Idea of the algorithm

Our algorithm is based on the surface tracking scheme and therefore, there are several limitations. A starting point must be determined and only one separated implicit surface can by polygonized for this first point. Several disjoint surfaces can be polygonized from a starting point for each of them. The whole algorithm consists of following steps:

1. Find a starting point $\mathbf{p}_0$.
2. Create the first triangle T_0, see Figure 21a.
3. Include the edges (e_0,e_1,e_2) of the first triangle T_0 into the active edges list.
4. Polygonize the first active edge *e* from the active edges list.
5. Delete the actual active edge *e* from the active edges list and include the new generated active edges to the end of the active edges list.
6. Check the distance between the new generated point $\mathbf{p}_{new}$ and all the other points lying on the border of already triangulated area (lying in all the other active edges).
7. If the active edges list is not empty return to step 4

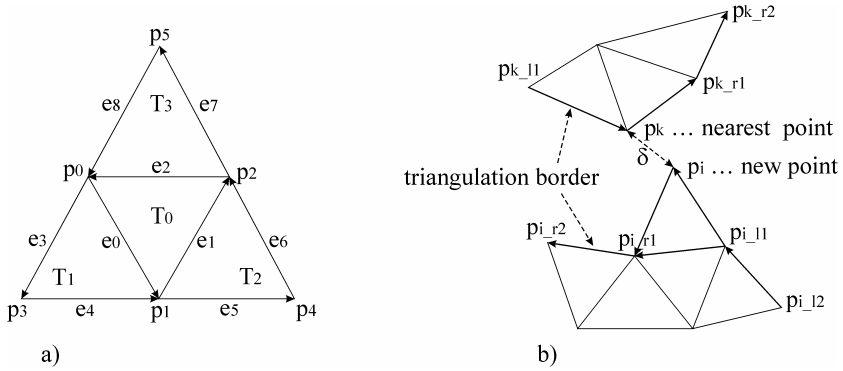

Figure 21. a) The first steps of the Edge spinning algorithm; b) Distance δ between the new point $\mathbf{p}_i$ and the nearest point $\mathbf{p}_k$.

Root finding

The algorithm looks for a new points' location by spinning of edges of already generated triangles. Usually, the polygonization algorithms seek points' coordinates following the gradient of an implicit function, [14]. Differential properties, [29], for each implicit function are different with the dependence on the modeling technique; therefore, the computing of a gradient of function f is influenced by a major error. Because of these reasons, in our approach, we have defined these restrictions for finding a new surface point $\mathbf{p}_{new}$:

- The new point $\mathbf{p}_{new}$ is sought in a constant distance, i.e. on a circle; then each new generated triangle preserves the desired accuracy of polygonization – the average edge's length δ_e. The circle radius is proportional to the δ_e.
- The circle lies in the plane defined by the normal vector of triangle T_{old} (see Figure 22) and axis o of the actual edge e; this guarantees that the new generated triangle is well shaped (isosceles).

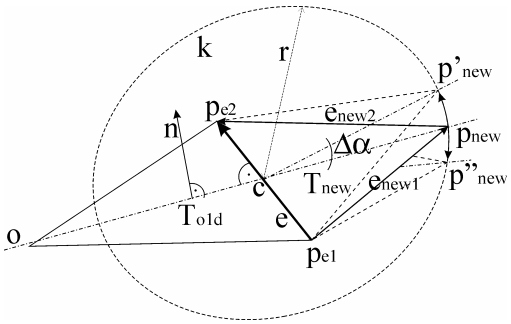

Figure 22. The root finding principle.

Then, the algorithm is as follows:

1. Set the point $\mathbf{p}_{new}$ to its initial position; the initial position is on the triangle's T_{old} plane on the other side of the edge e, see Figure 22. Let the angle of the initial position be $\alpha=0$.
2. Compute the functional values $f(\mathbf{p}_{new}) = f(\alpha)$, $f(\mathbf{p}'_{new}) = f(\alpha+\Delta\alpha)$ – initial position rotated by the angle $+\Delta\alpha$, $f(\mathbf{p}''_{new}) = f(\alpha-\Delta\alpha)$ - initial position rotated by the angle $-\Delta\alpha$; the rotation axis is the edge e.
3. Determine the right direction of rotation; if $|f(\alpha+\Delta\alpha)| < |f(\alpha)|$ then $+\Delta\alpha$ else $-\Delta\alpha$.
4. Let the functional values be $f_1 = f(\alpha)$ and $f_2 = f(\alpha\pm\Delta\alpha)$; actualize angle $\alpha = \alpha\pm\Delta\alpha$.
5. If $(f_1 \cdot f_2) < 0$ then compute the accurate coordinates of the new point $\mathbf{p}_{new}$ by the binary subdivision between the last two points corresponding to functional values f_1 and f_2; else return to step 4.
6. Check if both triangles T_{old} and T_{new} do not cross themselves; if the angle between these triangles $\beta > \beta_{lim}$ (see Figure 23) then point $\mathbf{p}_{new}$ is accepted; else point $\mathbf{p}_{new}$ is rejected and return to step 4.

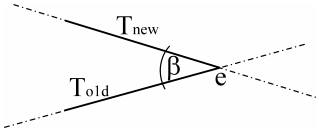

Figure 23. Angle between two triangles; the view is in direction of the edge's vector *e*.

Distance test

The Edge spinning algorithm marches over the surface of an implicit object. During this process, the new included point into the triangular mesh can lie near to the triangulation border (border of already triangulated area), see Figure 21b. The algorithm looks for the nearest point $\mathbf{p}_k$, to the new point $\mathbf{p}_i$, which lies on the triangulation border. If the distance δ is less then δ_{lim} then the algorithm has to solve this situation. The original algorithm checks distances with the algorithm complexity O(N), where *N* is a number of points on the triangulation border. This is the most time consuming part of the algorithm and its acceleration is suitable.

Acceleration

The original distance check algorithm takes more time if required scene details grow (growing number of points on the triangulation border). In case that the new included point can lie near to any point of the boundary, it is not possible to determine some subset of candidates to the nearest point ahead.

Advantageous solution is dividing of space into sub-spaces (sub-areas), similarly as in case of the Marching triangles algorithm above. The date structure of the point has to be extent of a pointer to its sub-area. Each sub-area contains its own list of incidence points, similarly as in Figure 19.

Then, the nearest point must lie in the same sub-area like the new included point or in the closest neighborhood. In order to validity of this theorem the next equation must be valid as well: $\sigma \geq \delta_{lim}$, where σ is the size of sub-areas (cube shape), see Figure 24, and δ_{lim} is the limit distance for distance check.

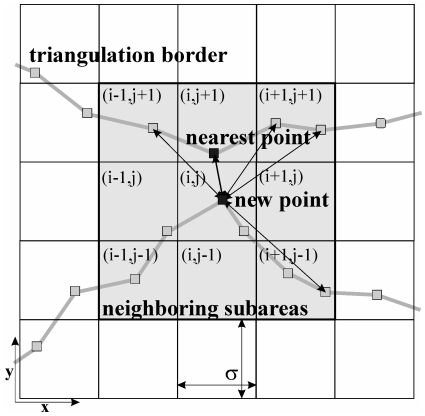

Figure 24. The space subdivision scheme for the Edge spinning algorithm.

The algorithm complexity of this solution is O(M), where *M* is a number of points in adjacent sub-areas and M<<N. Figure 24 shows 9 possible sub-areas in E^2 case, there are 27 possible sub-areas in E^3 case.

If we divide the computational time into two parts, the polygonization time and the distance check time, Figure 25 shows the percentage ratio of time with and without acceleration.

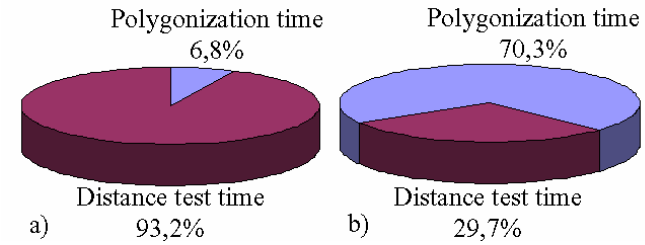

Figure 25. The time ratio between the Polygonization time and the Distance test time of the ES algorithm; a) without the space subdivision scheme, b) with the space subdivision.

The time needed for distance checking was significantly decreased. This acceleration technique is effective in result and simple for implementation. More examples are in the next section.

Experimental results

All the next experiments were accomplished on the implicit object Genus 3, Figure 26. Its implicit function is described as follows:

$$f(x,y,z) = r_z^4 \cdot z^2 - \left[1 - \left(\frac{x}{r_x}\right)^2 - \left(\frac{y}{r_y}\right)^2\right] \cdot \left[(x - x_1)^2 + y^2 - r_1^2\right] \cdot \left[(x + x_1)^2 + y^2 - r_1^2\right] = 0,$$

where r_x=6, r_y=3.5, r_z=4, r_1=1.2, x_1=3.9. (33)

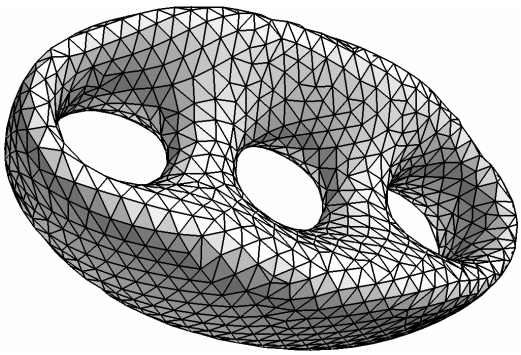

Figure 26. The implicit object Genus 3, 2 784 triangles, 1 388 vertices.

The implemented algorithms and methods have been included into the Modular Visualization Environment, [19], [27], developed by the Centre of Computer Graphics at University of West Bohemia.

Accelerated Marching triangles

Figure 27 shows the speed-up between the original MTR algorithm and the accelerated one. It can be seen that speed-up grows with the resolution linearly in the range of resolution used for experiments. The experiments proved that the proposed algorithm is especially convenient for cases where highly detailed objects are to be generated. Nevertheless, even for small resolution the proposed algorithm is significantly faster than the original one [14].

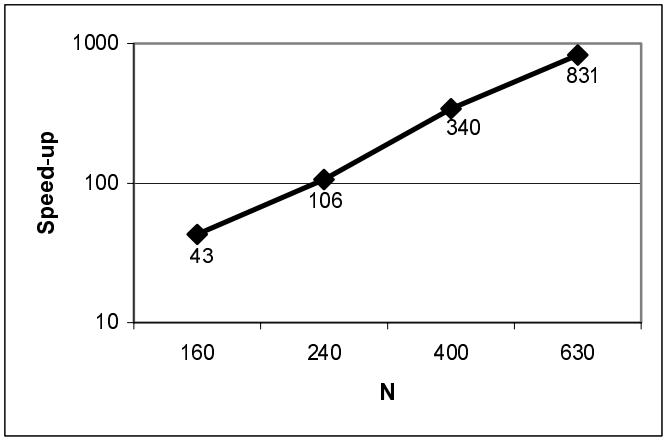

Figure 27. Speed-up between the original MTR algorithm and the accelerated version.

Table 1 contains values measured during this experiment. The next histogram, Figure 28, illustrates that the triangular mesh generated by the accelerated version of the MTR method consists of triangles with the similar shape properties.

	N	160	240	400	630
Original MTR algorithm	Triangles	15 535	34 945	97 785	244 295
	Vertices	7 763	17 468	48 886	122 143
	time [ms]	5 147	27 600	340 459	2 263 275
Accelerated MTR algorithm	Triangles	15 679	35 067	97 867	244 103
	Vertices	7 835	17 529	48 929	122 047
	time [ms]	120	260	1 001	2 724

Table 1. Values measured for both versions of the MTR algorithm.

The *N* variable, used in Figure 27 and in Table 1, represents a desired object detail. Specifically, the average triangles edges' length is proportional to *N* and to the computing area's size as well. With growing *N*, the triangles' edges are shorter, i.e. each side of the computing area is as though divided into *N* parts and length of such part is

the average edges' length of generated triangles. The computing area's size used in experiments is [<x_{min},x_{max}>,<y_{min},y_{max}>,<z_{min},z_{max}>]=[<-16,16>,<-16,16>,<-16,16>].

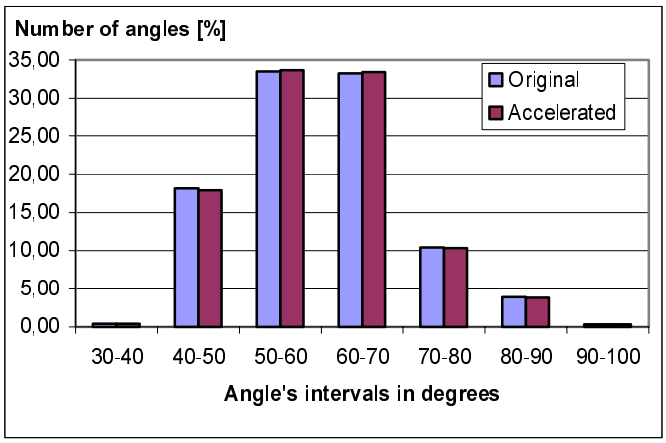

Figure 28. Histogram of the triangles' shape (angle distribution) for the MTR algorithms; generated for N=630, see Table 1.

Edge spinning algorithm

At first, we compare a number and shape of triangles generated by the Marching triangles [14], Edge spinning, and the Marching cubes [3] algorithms. For visual comparison, Figure 29 shows the Genus 3 object polygonized by these algorithms.

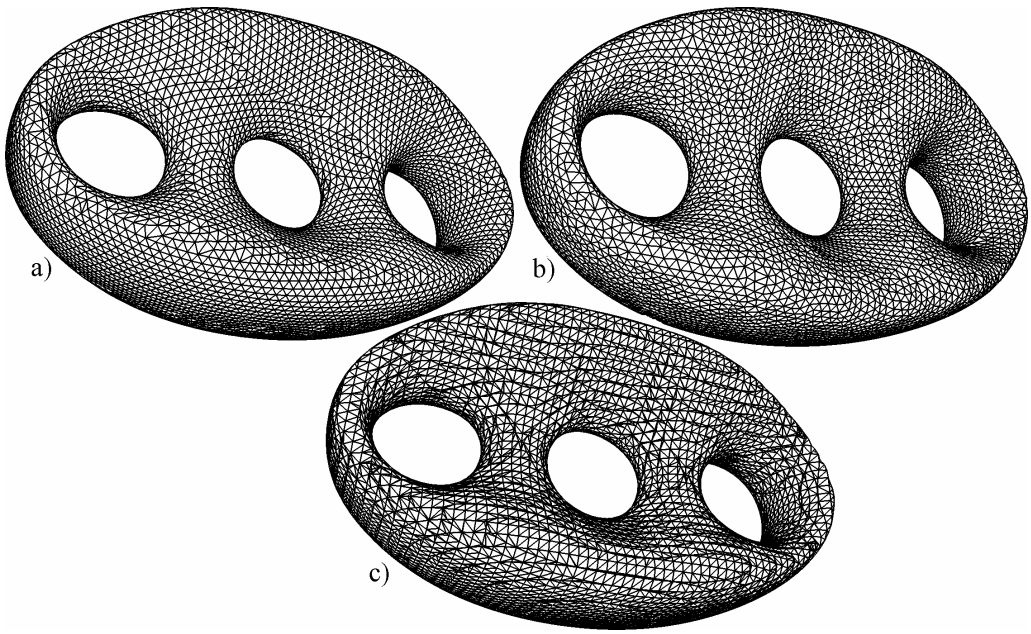

Figure 29. The Genus 3 object generated by a) the Edge spinning; b) the Marching triangles; c) by the Marching cubes algorithm.

The percentage ratio of the angles incidence is shown in Figure 30. This experiment demonstrates that the Edge spinning algorithm has the highest number of angles in

triangulation in interval <50º, 70º>. The Marching triangles method also generates well-shaped triangles and the Marching cubes algorithm generates poor polygonal mesh. The presented results were verified by using many nontrivial implicit surfaces.

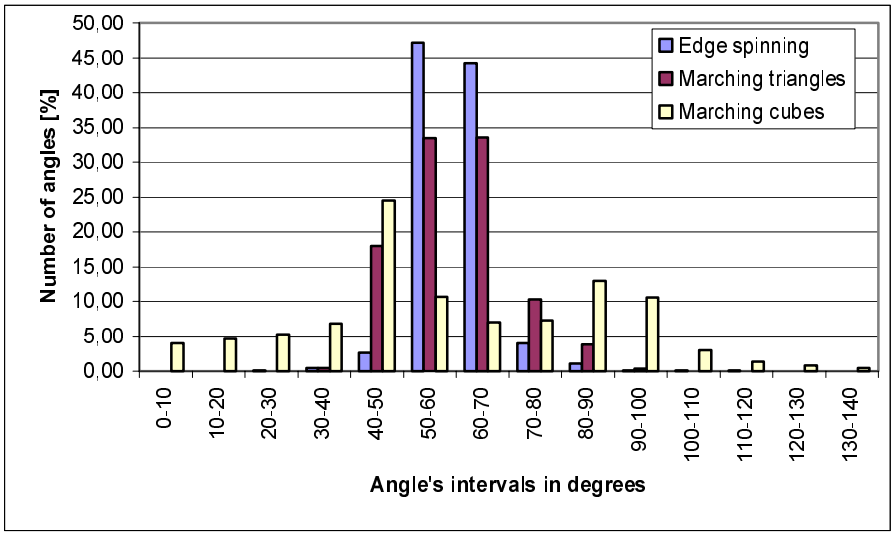

Figure 30. Histogram of angle distribution of triangular mesh, generated for N=1000, see Table 2.

Table 2 contains measured values from which follows that the Edge spinning algorithm generates about 15% triangles less then the Marching triangles algorithm and about 24% less then the Marching cubes algorithm.

	N	160	240	400	630	1000
Edge spinning	Triangles	13 368	29 892	81 708	207 290	521 320
	Vertices	6 680	14 942	40 850	103 641	260 656
Marching triangles	Triangles	15 689	35 267	97 943	244 107	613 641
	Vertices	7 840	17 629	48 967	122 049	306 816
Marching cubes	Triangles	17 568	39 520	109 608	271 344	684 016
	Vertices	8 772	19 756	54 800	135 668	342 004

Table 2. A number of triangles and vertices generated by mentioned algorithms.

Conclusion

We have presented a new fast modification of the Marching triangles algorithm, originally introduced in [14]. The used acceleration technique is an effective way to speed-up such type of geometric algorithms. The similarly good results were obtained by application to our new algorithm Edge spinning. This algorithm marches over the implicit object's surface and computes the accurate coordinates of new points by spinning the edges of already generated triangles. The ES method puts emphasis on the

shape of triangles of the resulting polygonal mesh but the presented algorithm can polygonize implicit surfaces which comply C^1 continuity. In future work, we want to modify the current algorithm for implicit functions with sharp features and with only C^0 continuity.

We have introduced the simple principle how to detect sharp edges in an implicit object. This approach was verified by the Marching triangles method and the obtained results proved its efficiency.

Chapter 4

Further research

All the polygonization algorithms developed so far have always had several disadvantages. The volume approaches (methods based on the Marching cubes and tetrahedra principle) usually generate polygonal meshes consisting of badly-shaped triangles. Due to the next processing, these meshes have to be further modified: improvement of the shape of triangles, reduction of the number of triangles in regions with a low curvature, etc. The reason why they are still in use is the fact that these methods are numerically stable. On the contrary, the surface approaches (methods based on the Marching triangles principle) generate well-shaped triangular meshes consisting of triangles shaped close to equilateral. These methods are limited by their high numerical sensitivity to the properties (continuity, differentiability, etc.) of the given implicit model.

In our previous work, we have verified and explored in the details the properties of the mentioned polygonization methods. In our further research, we will develop a hybrid algorithm that will take advantages of the both approaches (the volume based and the surface based). The algorithm will generate a triangular mesh locally adapted to the surface curvature. The whole polygonization algorithm will be controlled by the approximation error proportional to the distance of triangles from the real implicit surface, deviation of surface normals, surface curvature, etc. Our primary objective is the robustness of that algorithm and its independence on the concrete implicit object. The second objective is the polygonization speed, because the algorithm should be at least as fast as the volume approach with its surface refinement.

To sum up all the information given above, the aims of the doctoral thesis are as follows:

- to find a proper evaluation of the approximation error for controlling the polygonization process,
- to develop the algorithm solving implicit functions with only C^0 continuity,
- to minimize the approximation error in places with high curvature.

References

[1] Akkouche, S., Galin, E.: Adaptive Implicit Surface Polygonization using Marching Triangles, Computer Graphic Forum, 20(2): 67-80, 2001.

[2] Balsys, R.J., Suffern, K.G.: Visualisation of Implicit Surfaces, COMPUTERS & GRAPHICS 25, 89-107, 2001.

[3] Bloomenthal, J.: Graphics Gems IV, Academic Press, 1994.

[4] Bloomenthal, J.: Skeletal Design of Natural Forms, Ph.D. Thesis, 1995.

[5] Bloomenthal, J., Bajaj, Ch., Blinn, J., Cani-Gascuel, M-P., Rockwood, A., Wyvill, B., Wyvill, G.: Introduction to Implicit Surfaces, Morgan Kaufmann, 1997.

[6] Bloomenthal, J.: Implicit surfaces, Unchained Geometry, Seattle.

[7] Bloomenthal, J.: Polygonization of Implicit Surfaces, Computer Aided Geometric Design, vol. 5, no. 4, Nov. 1988, pp. 341-355.

[8] Bloomenthal, J., Ferguson, K.: Polygonization of Non-Manifold Implicit Surfaces, Computer Graphics, pp. 309-316 (Proc. SIGGRRAPH 95).

[9] Dekkers, D., Overveld, K., Golsteijn, R.: Combining CSG Modeling with Soft Blending using Lipschiz-based Implicit Surfaces, 1996.

[10] Faber, P., Fisher, R.B.: Pros and Cons of Euclidean Fitting, Proc. Annual German Symposium for Pattern Recognition (DAGM01, Munich), Springer LNCS 2191, Berlin, pp 414-420.

[11] Gomez, J., Velho, L.: Implicit Objects for Computer graphics, IMPA, 1998.

[12] Hart, J.C.: Ray Tracing Implicit Surfaces, SIGGRAPH 1993, pp. 1-16.

[13] Hart, J.C., Stander, B.T.: Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling, SIGGRAPH 1997, pp. 279-286.

[14] Hartmann, E.: A marching method for the triangulation of surfaces, The Visual Computer (14), pp.95-108, 1998.

[15] Hilton, A., Stoddart, A.J., Illingworth, J., Windeatt, T.: Marching Triangles: Range Image Fusion for Complex Object Modeling, International conf. On Image Processing, Lusanne, 1996, pp. 381-384.

[16] Hilton, A., Illingworth, J.: Marching Triangles: Delaunay Implicit Surface Triangulation, Computer Graphic Forum 20 (2) pp. 67–80, 2001.

[17] "Hyperfun: Language for F-Rep Geometric Modeling", http://cis.k.hosei.ac.jp/~F-rep/

[18] Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.-P.: Feature Sensitive Surface Extraction from volume data, SIGGRAPH 2001 proceedings.

[19] MVE – Modular Visualization Environment project, http://herakles.zcu.cz/research.php, University of West Bohemia in Plzeň, Czech Republic, 2001.

[20] Ning, P., Bloomenthal, J.: An Evaluation of Implicit Surface Tilers, IEEE Computer Graphics and Applications, vol. 13, no. 6, IEEE Comput. Soc. Press, Los Alamitos CA, Nov. 1993, pp. 33-41.
[21] Ohraje, Y., Belyaev, A.G., Pasko, A.: Dynamic Meshes for Accurate Polygonization of Implicit Surfaces with Sharp Features, Shape Modeling International 2001, IEEE, 74-81.
[22] Ohtake, Y., Belyaev, A.G.: Dual/Primal Mesh Optimization for Polygonized Implicit Surfaces, ACM Solid Modeling Symposium, Saarbrucken, Germany, ACM Press, 2002, pp. 171-178.
[23] Pasko, A., Adzhiev, V., Karakov, M., Savchenko, V.: Hybrid system architecture for volume modeling, Computer & Graphics 24 (67-68), 2000.
[24] Pasko, A., Adzhiev, V., Sourin, A., Savchenko, V.: Function Representation in Geometric Modeling: Concepts, Implementation and Applications, The Visual Computer, 8 (2), pp. 429-446, 1995.
[25] Pasko, A., Sourin, A., Savchenko, V.: Using Real Functions with Application to Hair Modeling, C&G(20), 1996, pp. 11-19.
[26] Rektorys, K.: Přehled užité matematiky, the textbook of mathematics (in Czech), SNTL 1981.
[27] Rousal, M., Skala, V.: Modular Visualization Environment - MVE, Int. Conf. ECI 2000, Herlany, Slovakia, pp.245-250, ISBN 80-88922-25-9.
[28] Rvachev, A.M.: Definition of R-functions, http://www.mit.edu/~maratr/rvachev/p1.htm
[29] Shapiro, V., Tsukanov, I.: Implicit Functions with Guaranteed Differential Properties, Solid Modeling, Ann Arbor, Michigan, 1999.
[30] Shu R., Zhou, Ch., Kankanhalli, M.S.: Adaptive Marching Cubes, The Visual Computer, 11: 202-217, 1995.
[31] Stoker, J.J.: Differential geometry, New York: Willey-Interscience, ISBN 0-471-50403-3, 1989.
[32] Takahashi, T., Yonekura, T.: Isosurface Construction from a Data Set sampled on a Face-Centered-Cubic Lattice, Int. Conf. ICCVG 2002, Zakopane, Poland, ISBN 839176830-9.
[33] Taubin, G.: Distance Approximations for Rasterizing Implicit Curves, ACM Transactions on Graphics, January 1994.
[34] Triquet, F., Meseure, F., Chaillou, Ch.: Fast Polygonization of Implicit Surfaces, WSCG 2001 Int.Conf., pp. 162, University of West Bohemia in Pilsen, 2001.
[35] Uhlíř, K., Skala, V.: Kompilovaný HyperFun, Research report (in Czech) No. DCSE/TR-2002-07, University of West Bohemia, 2002.
[36] Velho, L.: Simple and Efficient Polygonization of Implicit Surfaces, Journal of Graphics Tools, Volume 1 Number 1, 1996, pp. 5-24.
[37] Velho, L., Gomes, J., Figueiredo, L.H.: Implicit Objects in Computer Graphics, Springer, ISBN 0-387-98424-0, 2002.
[38] Wyvill, B., Guy, A., Galin, E.: Extending the CSG Tree Warping, Blending and Boolean Operations in an Implicit Surface Modeling System, Computer Graphics Forum, 18(2), 149-158, June 1999.
[39] Wyvill, B., Bloomenthal, J.: Interactive Techniques for Implicit Modeling, Symposium on Interactive 3D Computer Graphics, Snowbird, UT, in Computer Graphics, 24, 2, Mar. 1990, pp. 109-116.

[40] Wyvill, B., Overveld, K.: Polygonization of Implicit Surfaces with Constructive Solid Geometry, Journal of Shape Modeling, vol. 2, no. 4, World Scientific Publishing, 1997, pp. 257-273.
[41] Wyvill, B., Jepp, P., Overveld, K., Wyvill, G.: Subdivision Surfaces for fast Approximate Implicit Polygonization, University of Calgary, Dept. of Computer Science, Research Report 2000-671-23, 2000.

Appendix A

Publications

[i] Čermák, M., Skala, V.: Space Subdivision for Fast Polygonization of Implicit Surfaces. The Fifth International Scientific Conference, ECI 2002, Slovakia, ISBN 80-7099-879-2, pp. 302-307, October 10-11, 2002.

[ii] Čermák, M., Skala, V.: Polygonization by the Edge Spinning. 16th Conference on Scientific Computing, Algoritmy 2002, Slovakia, ISBN 80-227-1750-9, pp. 245-252, September 8-13, 2002.

[iii] Čermák, M., Skala, V.: Accelerated Edge Spinning Algorithm for Implicit Surfaces. International Conference on Computer Vision and Graphics, ICCVG 2002, Poland, ISBN 839176830-9, pp. 174-179, September 25-29, 2002.

[iv] Čermák, M., Skala, V.(supervisor): Visualization of Scenes which are Defined by Implicit Functions and CSG trees, MSc. Thesis (in Czech), University of West Bohemia in Pilsen, 2001.

Accepted for publication:

[v] Čermák, M., Skala, V.: Edge Spinning Algorithm for Implicit Surfaces. Accepted for publication in journal 'Mathematics and Computers in Simulation', IMACS/Elsevier Science B.V., Roma, Italy.

Citation:

[vi] Uhlíř, K., Skala, V.: The Implicit Function Modeling System – Comparison of C++ and C# Solutions, C# and .NET Technologies' 2003, Workshop proceedings, pp. 87-92, February 5.-8., 2003.

Appendix B

Stays and Lectures Abroad

Stays:

15.2.2000 – 15.6.2000	University of Lisboa, Portugal, Erasmus/Socrates project

Conferences:

1.9.2002 – 6.9.2002	Eurographics 2002, Saarbrücken, Germany
8.9.2002 – 13.9.2002	Algoritmy 2002, Podbanske, Slovakia, [ii]
25.9.2002 – 29.9.2002	ICCVG 2002, Zakopane, Poland, [iii]
10.10.2002 – 11.10.2002	ECI 2002, Herlany, Slovakia, [i]