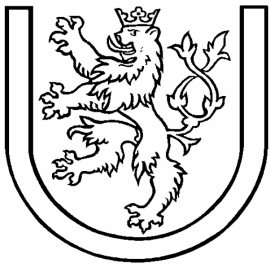
University of West Bohemia in Pilsen
Faculty of Applied Sciences
Department of Computer Science and Engineering



Dissertation thesis

Methods for Implicit Surfaces Polygonization

Martin Čermák
e-mail: cermakm@kiv.zcu.cz
URL: http://herakles.zcu.cz/~cermakm

Supervisor: prof. Ing. Václav Skala, CSc. Plzeň, November 2004

Abstrakt

Modelování objektů a jejich zobrazování patří k základním úkolům počítačové grafiky. V posledních letech se stalo atraktivním modelování objektů pomocí implicitních funkcí. Implicitní modelování je často užíváno v modelovacích programech založených na CSG stromech hlavně z důvodu, že implicitní funkce přímo definují objemová data. Zobrazování takto definovaných objektů je možné buďto algoritmy založenými na principu sledování paprsku popř. aproximačními algoritmy, které převádějí implicitní reprezentaci objektů na polygonální resp. trojúhelníkové sítě. Takový aproximační proces je obvykle nazýván polygonizací implicitní funkce. Zobrazování trojúhelníkových sítí je podporováno naprostou většinou běžných grafických akcelerátorů, tj. jedná se o současný standard. Práce s trojúhelníkovými sítěmi je rychlá a, narozdíl od metod sledování paprsku, je možné provádět libovolné pohledové transformace s objekty bez nutnosti opakovaného vyvolání zobrazovacího algoritmu. Trojúhelníkové sítě jsou také podporovány profesionálními 3D modelovacími programy, tj. není nic snazšího, než importovat výsledný polygonální model do vaší oblíbené aplikace a využít pro jeho další zpracování již profesionálních nástrojů.

Prezentovaná disertační práce je rozčleněna do tří základních kapitol. V úvodu je čtenář seznámen se základními pojmy nutnými ke správnému porozumění probírané problematice. Další kapitola obsahuje podrobný výklad pojmu polygonizace společně s různými způsoby, jak vyhodnocovat aproximační chybu. Zde jsou také představeny základní algoritmy určené pro polygonizaci implicitních funkcí. Jednotlivé metody jsou rozděleny podle svých vlastností na objemové/povrchové přístupy, adaptivní/neadaptivní, s/bez schopnosti polygonizovat implicitně definované scény sestávající z více objektů. V další kapitole je čtenář seznámen s algoritmy navrženými autorem předkládané práce. Pořadí, ve kterém jsou jednotlivé metody představovány, je dáno pořadím v jakém byly vyvinuty. V každé podkapitole je konkrétní algoritmus vysvětlen, zhodnocen a jsou prezentovány výsledky dosažené během jeho testování. V závěru práce je zhodnocen vědecký přínos autorova díla a jsou nastíněny možné směry dalšího výzkumu.

Klíčovou částí disertační práce je adaptivní algoritmus navržený autorem, jehož výklad je situován v druhé polovině práce. Prezentovaný algoritmus adaptivně polygonizuje implicitní objekty s definovanou přesností a navíc je schopen zpracovat scény sestávající z více těles. Algoritmus byl publikován na několika mezinárodních konferencích pod pracovním názvem *adaptive edge spinning algorithm*.

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 thesis is divided into three main chapters. An introduction contains fundamental notions necessary for a reader to proper understanding of problems presented. In the next chapter, polygonization term is described in detail including definition of varied techniques how to measure an error of a polygonal approximation. There are also several basic algorithms used for polygonization of implicit surfaces introduced in this chapter. The presented methods are separated according to their properties into volume/surface approaches, adaptive/non-adaptive methods, able/non-able to polygonize scenes consisting of more disjoint implicit surfaces, etc. The next chapter presents algorithms and new approaches proposed by the author of this thesis. The algorithms are introduced in a chronological order of their developing. In each section, a principle of a new algorithm is explained and concluded as well as experimental results. The closing part of the thesis contains conclusion of the author's contribution and an outline of possible future work as well.

Key part of the thesis is an adaptive approach developed by the author that is situated in second half of the work. The presented approach is able to polygonize unknown implicit scenes with defined accuracy as well as scenes consisting of more disjoint surfaces. Its working name is *adaptive edge spinning algorithm* and it has been published on several international conferences.

Acknowledgement

First of all I would particularly like to thank my supervisor, prof. Václav Skala, for his support and guidance during my PhD study.

Many special thanks go to the other members of our computer graphics group at University of West Bohemia in Pilsen.

Not least, I would like to thank my family and my girlfriend Jana for their patience and encouragement throughout my studies.

The work was supported by the Ministry of Education of the Czech Republic - project MSM 235200005.

Contents

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(**p**) = 0, where **p** = [x,y,z] is a point in three-dimensional Euclidean space. An iso-surface is a similar set of points for which f(**p**) = c, where *c* is the iso-contour value of the surface. An implicit surface *S* is characterized as a set of points whose potential f(x,y,z) equals a threshold value denoted by *T*. More precise mathematical definition is described in [5] or [43].

$$S = \{M(x,y,z) \in E^3 \mid f(x,y,z) \in T\}. \quad (1)$$

There are two different definitions for implicit objects. The first one [5], [7], [8] defines an implicit object as f(**p**) < 0 and the second one, F-rep (functional representation) [22], [34], [41], defines it as f(**p**) ≥ 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 [17], which generates excellent photo-realistic images of implicit objects. Alternatively, an image of the function can be created with volume rendering.

1.1. 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, [9]. The three-dimensional case is:

trigonometric	$x = (\cos(\alpha)\cos(\beta), y = \sin(\alpha), z = \cos(\alpha)\sin(\beta), \alpha \in [0,\pi], \beta \in [0,2\pi)$	
rational	$x = 4st/w, y = 2t(1-s^2)/w, z = (1-t^2)(1+s^2)/w,$ for $w = (1+s^2)(1+t^2), s,t \in [0, 1]$	
implicit	$f(x,y,z) = x^2+y^2+z^2-1$	(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, [9].

1.2. 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, [9].

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), [32]. 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.

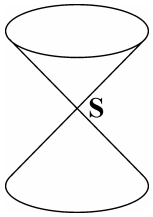


Figure 1. The apex of a cone is a singular point.

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*, [43].

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 [9].

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', [9].

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.

1.3. Surface curvature

1.3.1. 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:

$$Hf(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \mathbf{L} & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \mathbb{M} & \mathbf{O} & \mathbb{M} \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \mathbf{L} & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}. \tag{6}$$

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, [43].

1.3.2. 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 E^{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.

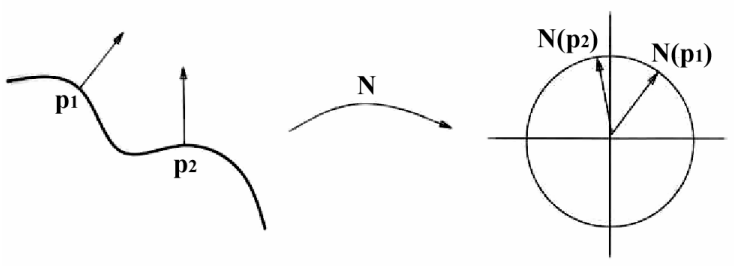


Figure 3. The Gauss map, taken from [43].

Note that a submanifold is defined as follows, see [43] for details. A submanifold S of another manifold M is a subset $S \subset M$ with a manifold structure, such that the inclusion map $i:S \to M$, $i(\mathbf{p})=\mathbf{p}$, is an embedding. If $S^n \subset M^m$ is an n-dimensional submanifold of an m-dimensional manifold M, with $n \leq m$, then the difference $k = m - n$ is called the *codimension* of S in M.

The derivative N' of N (along its arguments x, y, z) 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, [43]. It is easy to show that:

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

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

1.3.3. 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 $\mathbf{p}$ 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 $\mathbf{p}$.

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

A surface is completely determined up to rigid motion by its first and second fundamentals forms, [32], [37].

If $M = f^{-1}(c)$ is a regular implicit surface in E^{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:

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

1.3.4. Surface curvature

The second fundamental form allows us to investigate the curvature of a surface. The *Normal curvature* of *M* at **p** in the direction **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(**v**) is equal to the normal component of acceleration of any curve, contained in *M*, passing through **p** with velocity **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 **p** and the correspondent unit eigenvectors of $N'(\mathbf{p})$ are called *principal directions*. The principal curvatures are stationary values of normal curvature k(**p**) and among them k(**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*, [43].

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

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

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 **p** is 1/n times the trace of S(**p**):

$$\overline{K}(\mathbf{p}) = \frac{1}{n}\operatorname{trace} S(\mathbf{p}) = \frac{1}{n}\sum_i^n k_i(\mathbf{p}) \tag{14}$$

It is the average value of the principal curvatures at **p**.

The determinant of S(**p**) is called the *Gauss-Kronecker curvature* K_G of *M* at **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.

1.4. Modeling of Implicit objects

1.4.1. 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, [48]. These operations may create hard-edged functions that conventional polygonization algorithms cannot accurately approximate, see Figure 4.

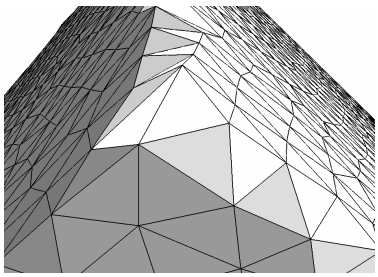


Figure 4. A corner of a cube modeled as intersection of six half-spaces.

The exact analytical definitions of the set-theoretic operations of functionally described objects have been proposed in the theory of R-functions, [34], 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 $-1 < \alpha(f_1,f_2) \leq 1$, $\alpha(f_1,f_2) = \alpha(f_2,f_1) = \alpha(-f_1,f_2) = \alpha(f_1,-f_2)$.

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:

$$f_1 \mid f_2 = \max(f_1,f_2)$$

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

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:

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

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:

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

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 [12], [16], [30], [31], [43].

1.4.2. 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, [7].

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, [7]:

- 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.

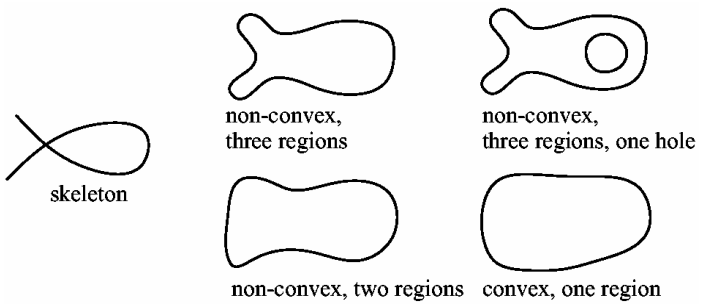


Figure 5. A skeleton and possible resulting surfaces, taken from [7]**.**

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, [46], [44], [47] when a designer creates a shape by interactively defining the skeleton and various parameters that control how the skeleton becomes a polygonized surface.

2. Polygonization

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*, [35], this is always possible because all manifold surfaces may be triangulated. Such mesh conversion is popularly known as *polygonization*.

In this chapter, an overview of existing polygonization algorithms is presented as well as some techniques for measurement of approximation quality.

2.1. Approximation error

The approximation error is a measure of difference between the polygonal model and its mathematical description. In available papers, there is not emphasis on measurement of the error. Therefore, the author has proposed several possibilities how to evaluate this difference. These techniques are used for comparison of algorithms properties in following sections.

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

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

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

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

where $\mathbf{p}_i$ 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:

$$E_{at} = \frac{\sum_{i=1}^{M} \mathrm{dist}(\mathbf{c}_{ti})}{M}, \tag{22}$$

where $\mathbf{c}_{ti}$ is the centre of gravity of an i^{th} triangle and *M* is a number of triangles.

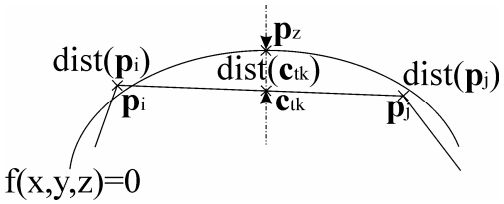


Figure 6. Approximation of an implicit object with a triangular net (contours and lines in two-dimensional example); distance of point $\mathbf{c}_{tk}$ (the centre of gravity of a triangle) from the real surface.

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:

$$\mathrm{dist}_A(\mathbf{p}) = |f(\mathbf{p})| \tag{23}$$

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*, [39], defined as follows:

$$\mathrm{dist}_T(\mathbf{p}) = \frac{|f(\mathbf{p})|}{\|\nabla f(\mathbf{p})\|} . \tag{26}$$

The Taubin's distance is the first order approximation to the exact distance, but the approximate distance is also biased in some sense, [13]. If, for instance, a data point **p** is close to a critical point of the function, i.e., $\|\nabla f(\mathbf{p})\| \approx 0$, but $f(\mathbf{p}) \neq 0$, the distance becomes large which is certainly a limitation.

There is also another point of view how to measure approximation quality than investigation of a distance from the real surface. Such approaches are proportional to surface curvature estimation. Curvature error along an edge can be expressed as a deviation of surface normal vectors at points of the edge, see Figure 7. Such curvature error is called the *Angle error* of an edge.

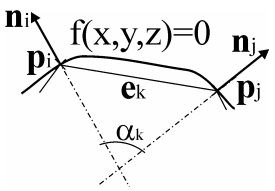


Figure 7. Curvature error of the edge e_k is measured as deviation α_k between surface normals $\mathbf{n}_i$ and $\mathbf{n}_j$.

The average angle error could be then determined by the following simple formula:

$$E_{angle} = \frac{\sum_{k=1}^{P} a_k}{Q} , \tag{27}$$

where *Q* is a number of edges in triangulation.

Another curvature error could be measured for a triangle as a deviation between the triangle's normal vector and the real normal vector of the implicit function determined at the centroid of the triangle, see Figure 8. The second curvature error is called the *Centroid angle error* of a triangle.

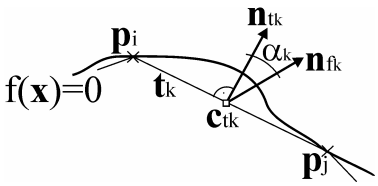


Figure 8. Curvature error of the triangle t_k is measured at centroid $\mathbf{c}_{tk}$ as deviation α_k between the triangle's normal vector $\mathbf{n}_{tk}$ and the real normal vector of the implicit function $\mathbf{n}_{fk}$.

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

$$E_{c\,angle} = \frac{\sum_{k=1}^{M} a_k}{M}, \tag{28}$$

where *M* is a number of triangles.

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:

$$S_p = \sum_{ti=1}^{M} S_{ti}, \tag{29}$$

where S_{ti} is the surface area of the i^{th} triangle and *M* is a number of triangles.

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

$$E_s = \left| 1 - \frac{S_p}{S_r} \right|, \tag{30}$$

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

2.2. Triangulation quality

A quality of shape of triangles generated is usually measured by the following criterions:

- Edge length criterion – a ratio between the shortest and the longest edge of a triangle
- Angle criterion – a ratio between the smallest to the biggest angle of a triangle
- Histogram of angles distribution – a number (or a percentage) of angles in given intervals

2.3. 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 **p** is usually computed using linear interpolation:

$$\mathbf{p} = \alpha c_1 + (1-\alpha)c_2, \text{ where } \alpha = f(c_2)/\big(f(c_2) - f(c_1)\big), f(c_1), f(c_2) \neq 0 \tag{31}$$

This method is popularly known as ‘marching cubes’ or ‘marching tetrahedra’. The standard Marching cubes (MC) and the Marching tetrahedra (MTE) algorithms [5], [7] 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 9a shows the basic 15 configurations of a cube and the configurations of a tetrahedron are shown in Figure 9b.

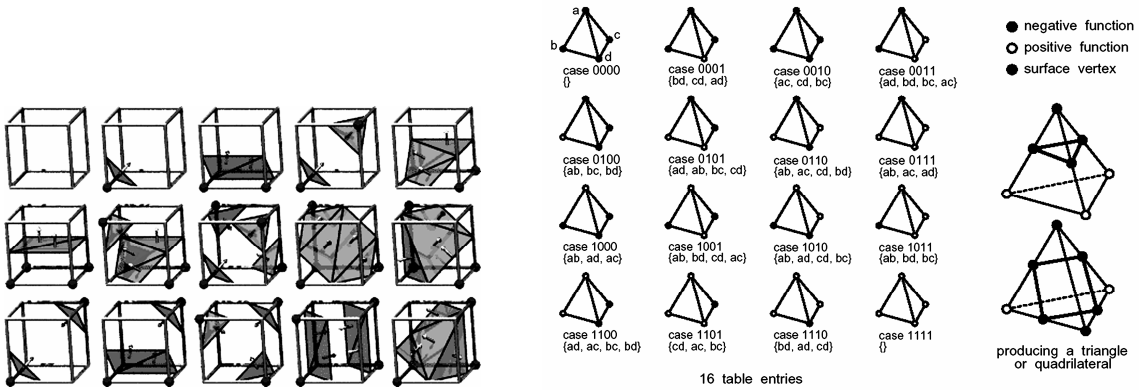


Figure 9. a) The basic 15 configuration of a cube, b) the configuration of a tetrahedron, taken from [5].

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.

2.4. Piecewise-Linear continuation

Piecewise-linear principles have been applied to implicit surfaces using a tetrahedral cell and a cubic cell, [5], [40]. 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, [5], [7]), see Figure 10. In comparison, exhaustive enumeration requires $O(N^3)$ samples. Compared with subdivision, continuation appears less prone to under-sampling.

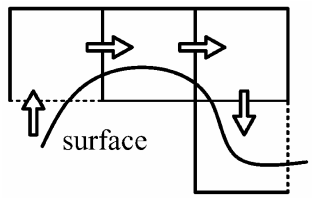


Figure 10. Continuation scheme, 2D example for illustration, taken from [5].

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.

2.5. Predictor-Corrector continuation

Predictor-corrector methods [1], [19], [20], [21] 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.

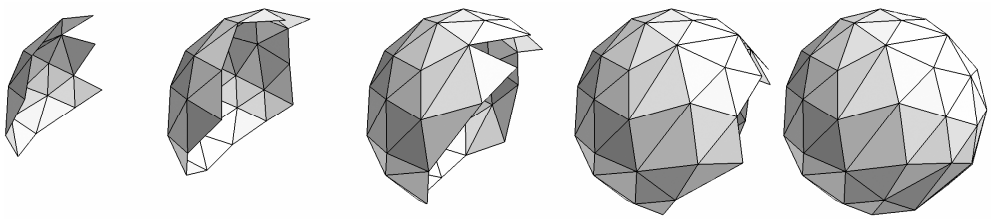


Figure 11. Continuation scheme, new triangles are directly generated on an implicit surface.

2.5.1. Marching triangles

The idea of the Marching triangles (MTR) algorithm, [19], consists of five steps:

Algorithm 1. Marching triangles principle.

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 12a). 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] $\prod_0$.

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 12b).

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 13a).

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 13b). Delete $\prod_m$.

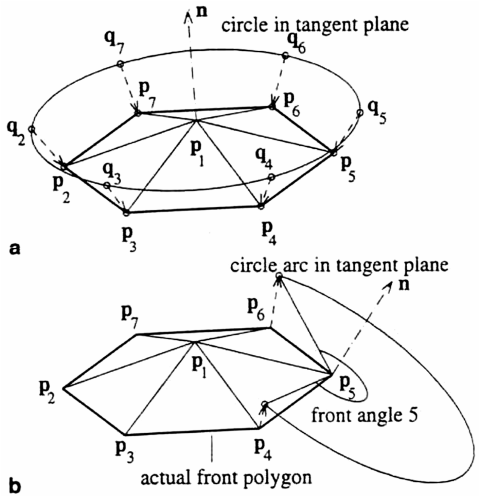


Figure 12. The first steps of the Marching triangles algorithm, taken from [19].

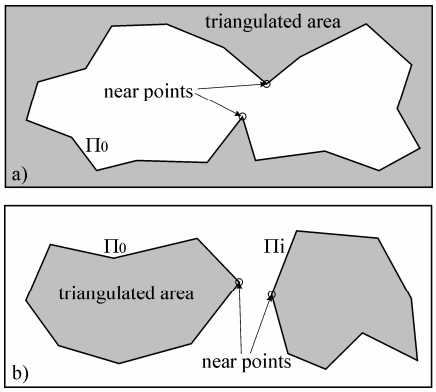


Figure 13. (a) Dividing the actual front polygon (step 2a of the MTR algorithm) and (b) uniting two front polygons (step 2b of the algorithm).

[1] the border of the triangulation

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.

2.6. 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], [10], [36]. Accurate representation of non-differentiable *f*, however, may require explicit computation of its singular points.

2.6.1. 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 [10], 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, [18], 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 14 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 14 right, then the face neighbor should be subdivided. Without such subdivision, a hole will appear in the surface.

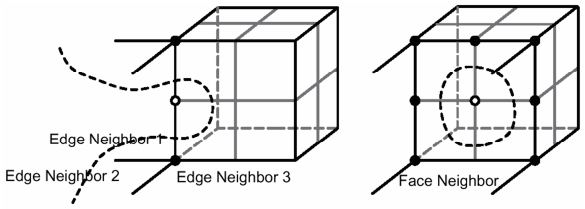


Figure 14. Conditions warranting subdivision of adjacent cubes; midpoint of an edge (left) and midpoint of a face (right), taken from [10].

The generalized cylinder in Figure 15 was created by this adaptive algorithm.

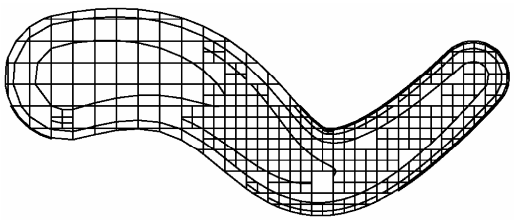


Figure 15. Adaptively subdivided generalized cylinder, taken from [10].

2.6.2. 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 [20] 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 16).

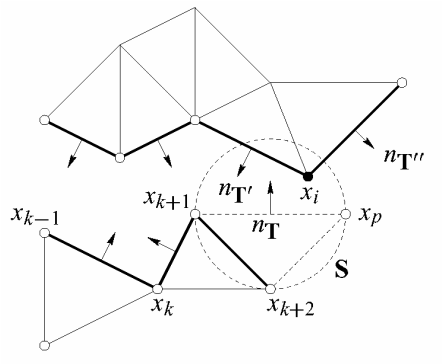


Figure 16. Creation of a new triangle T: the empty sphere criterion does not apply as the sphere *S* intersect another part of the mesh (at vertex $\mathbf{x}_i$) whose surface normals $\mathbf{n}_{T'}$ and $\mathbf{n}_{T''}$ exhibits a different orientation than $\mathbf{n}_T$. The picture is taken from [1].

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

Algorithm 2. Adaptive Marching triangles.

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 [20] 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 17. This step creates a new point on the surface denoted as $\mathbf{x}_s$.

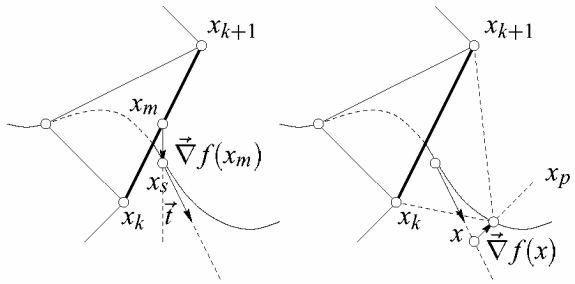


Figure 17. Characterization of the surface point $\mathbf{x}_s$ and the projected point $\mathbf{x}_p$, taken from [1].

2. Computation of the starting point. Let **t** be the unit tangent vector (see Figure 17) to the surface at the surface vertex position $\mathbf{x}_s$ defined as:

$$\mathbf{t} = \frac{\mathbf{e}_k \times \nabla f(\mathbf{x}_s)}{\left\| \mathbf{e}_k \times \nabla f(\mathbf{x}_s) \right\|}. \tag{32}$$

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

$$d = \frac{\sqrt{3}}{2}\bar{e}, \text{ where } \bar{e} = \frac{\left\|\mathbf{e}_{k-1}\right\| + \left\|\mathbf{e}_k\right\| + \left\|\mathbf{e}_{k+1}\right\|}{3}, \tag{33}$$

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 $\nabla f(\mathbf{x})$.

$$\mathbf{y} = \mathbf{x} - \alpha \frac{f(\mathbf{x}) \nabla f(\mathbf{x})}{\left\| \nabla f(\mathbf{x}) \right\|^2}, \tag{34}$$

        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**].

2.7. Surface refinement

One possible solution for polygonization of implicit object with sharp features is refinement of an initial triangular mesh, [4], [24], [27], [28], [42]. 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.

The algorithm introduced in [28] 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.

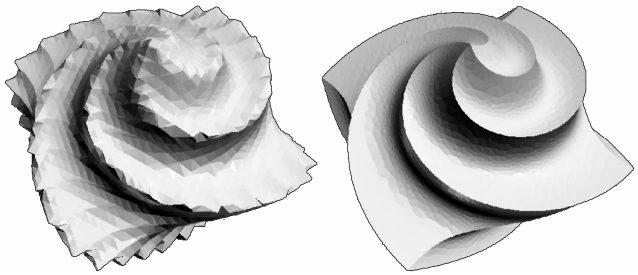


Figure 18. The initial mesh (left) created by the Marching cubes algorithm and its optimized version (right), taken from [28].

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.

2.8. Particle systems

Particle systems (physically based techniques) start from initial positions in space and seek their equilibrium positions, i.e. positions where a potential function | f | is minimal – on an implicit surface, [14], [15]. The desired polygonal approximation is then obtained by computing the Delaunay triangulation associated with the points.

The interpretation of the gradient of | f | as a force field implies the following equation of motion for a unit mass particle:

$$\frac{d^2x}{dt^2} + g\,\frac{dx}{dt} + sign(f)\nabla f = 0\,, \tag{35}$$

where γ is a positive real number representing friction proportional to velocity, [14].

Figure 19a shows the trajectories of a particle system associated with a two-dimensional curve with 2 components and Figure 19b shows the final equilibrium positions of these particles along the curve.

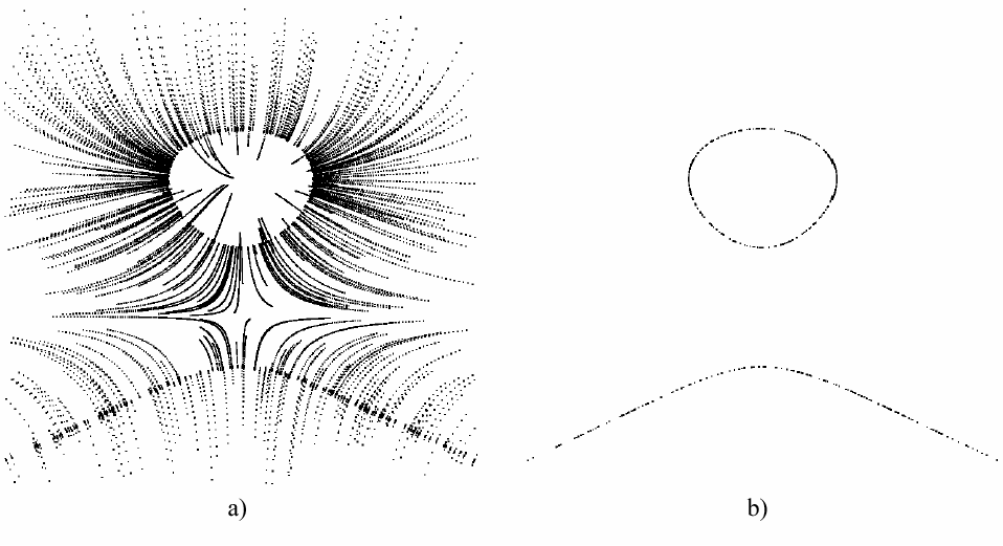


Figure 19. Trajectories (a) and final positions (b) of particles for 2D curve, taken from [14].

Figure 20a shows the sample points on the surface of a sphere and Figure 20b shows the polygonal approximation for the sphere.

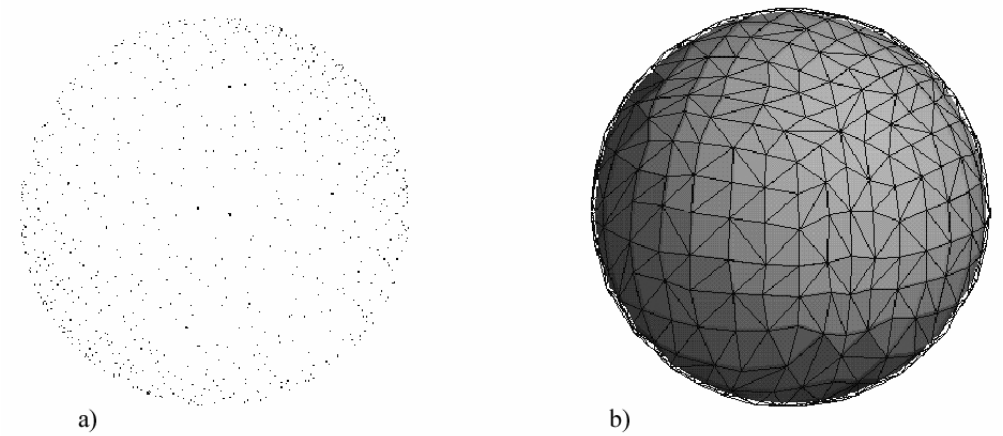


Figure 20. Sample points on the surface of a sphere (a) and polygonization of the sphere (b), taken from [14].

2.9. 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 [11].

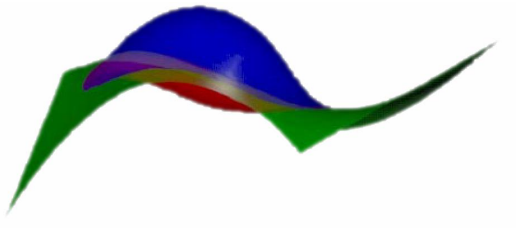


Figure 21. Polygonized non-manifold, taken from [11].

3. Novel polygonization approaches

In previous sections, an overview about implicit surfaces, their modeling and polygonization has been introduced. The author's work will be presented in the following text. This chapter starts with modifications of the Marching triangles algorithm that has been mentioned in section 2.5.1. The following section contains an introduction of the new polygonization method - Edge spinning (ES). The first version of the algorithm is non-adaptive and its adaptive modification is presented next. The ES algorithm is based on the similar principle as the Marching triangles method. The ES approach has been implemented in a standard way as well as using an acceleration technique. Thereinafter, the ES algorithm has been modified to be able to polygonize implicit object with sharp features. A modification of the algorithm, to be able to triangulate implicit scenes consisting of more disjoint surfaces, is presented finally.

Note that sequence of the following sections is chronological according to development of the algorithms. In each section, the comparison and experimental results of the given method are demonstrated. All experiments have been realized on a computer Athlon XP 2500+, 512MB DDR400.

3.1. Marching triangles improvement

The original algorithm described in [19] 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 Algorithm 1). Our modification of the algorithm is directed precisely towards achieving this step.

3.1.1. Decreasing the algorithm complexity

The first distance check (FDC, step 2a of Algorithm 1) is of complexity:

$$O\left(\tfrac{1}{2} m * (m-5)\right) \Rightarrow O\left(m^2\right), \tag{36}$$

where m is a number of points in the actual front polygon.

The second distance check (SDC, step 2b of Algorithm 1) is of complexity:

$$O(m * u), \tag{37}$$

where u is a number of points in all the other front polygons.

Both distance checks, FDC and SDC, are evaluated at each step of the MTR algorithm and, therefore, the final algorithm complexity is:

$$O\left(\left(m^2 + m * u\right) * s\right) \Rightarrow O\left(m * \left(m + u\right) * s\right) \approx O\left(N^3\right), \tag{38}$$

where *s* is a number of repetitions of the MTR algorithm.

There is good to realize that the shape and structure of the actual front polygon is only modified during the polygonization process and all the other front polygons stand without any changes. Thereinafter, the actual front polygon's shape is only modified in one location where new points are included. The result of those causes is: the main part of a scene is static and only one local limited area is dynamically modified. Therefore, the distance checking need only be performed for new points and Step 2 of the MTR algorithm can be written as follows.

Step 2: Check if only the **new** points $\boldsymbol{p_i}$ of the actual front polygon are near … (both steps *a* and *b* are without changes).

This means that the algorithm complexity for one step and one inserted point is decreased and can be expressed for FDC as:

$$O\left(m\right), \tag{39}$$

where *m* is a number of points in the actual front polygon, and for SDC as:

$$O\left(u\right), \tag{40}$$

where *u* is a number of points in all the other front polygons.

Therefore, the final algorithm complexity is:

$$O\left(\left(m + u\right) * s\right) \approx O\left(N^2\right), \tag{41}$$

where *s* is a number of repetitions of the MTR algorithm.

The algorithm complexity of the MTR method was reduced by one level. Therefore, the usage of the algorithm for polygonization of more detailed objects (with a large number of polygons as its output) was significantly increased. Nevertheless, our experiments proved that the MC algorithm (surface tracking approach) is, for polygonization of highly detailed objects, significantly speedier than this modified MTR method. The reason is that the distance checking is still using large algorithm complexity for the growing number of points in front polygons.

3.1.2. Acceleration

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 [19]). 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 22 shows a distance check for a new included point (for illustration only the E^2 example).

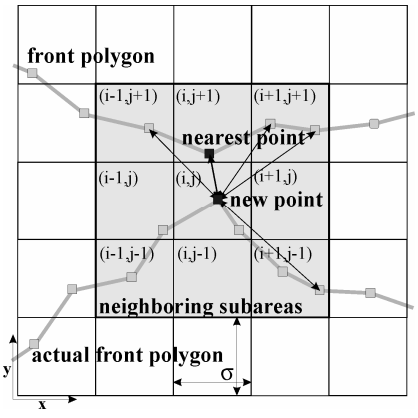


Figure 22. Space subdivision scheme.

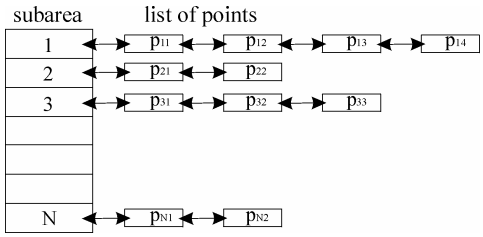


Figure 23. The data structure for the space subdivision scheme.

Each sub-area contains a list of front polygons' points located inside. The data structure used for the subdivision scheme is shown in Figure 23. 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.

3.1.3. Edge detection

Detection of sharp edges is a modification of the MTR method (section 2.5.1) at the step of finding location of a new point (in step 3 of the MTR algorithm, see Algorithm 1). The principle of the algorithm is in knowledge of normal vectors at 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.

Algorithm 3. Edge detection for the Marching triangles method.

1. Initialization, **a** = **p**, **b** = **q**, $\mathbf{n}_a$ … normal vector in the point **a**, $\mathbf{n}_b$ … normal vector in the point **b**.
2. **c** = 0.5*(**a**+**b**) … binary subdivision between the points **a**, **b**.
3. Let the normal vector at the point **c** be $\mathbf{n}_c$, and α be the angle between vectors $\mathbf{n}_a$ and $\mathbf{n}_c$.
4. If $\alpha > \alpha_{lim}$ then **b** = **c** else **a** = **c**.
5. If the distance between points **a**, **b** is less then some ε, the desired point is **b**, else return to step 2.

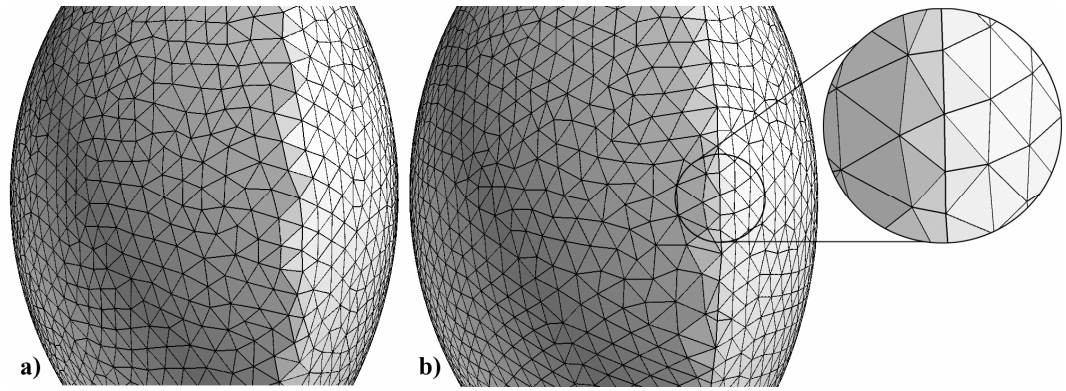


Figure 24. The implicit object modeled as intersection of two spheres; polygonized a) without edge detection; b) with edge detection algorithm.

3.1.4. Experimental results

The next experiments were accomplished on the implicit object Genus 3, Figure 25. Its implicit function is described as follows:

$$f(\mathbf{x}) = r_z^4 \cdot z^2 - \left[1 - (x/r_x)^2 - (y/r_y)^2\right] \cdot \left[(x - x_1)^2 + y^2 - r_1^2\right] \cdot \left[(x + x_1)^2 + y^2 - r_1^2\right] = 0, \qquad (42)$$

where the parameters are r_x=6, r_y=3.5, r_z=4, r_1=1.2, x_1=3.9.

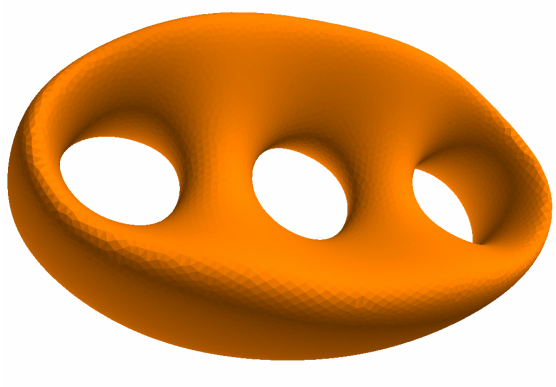


Figure 25. The implicit object Genus 3.

Figure 26 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 [19].

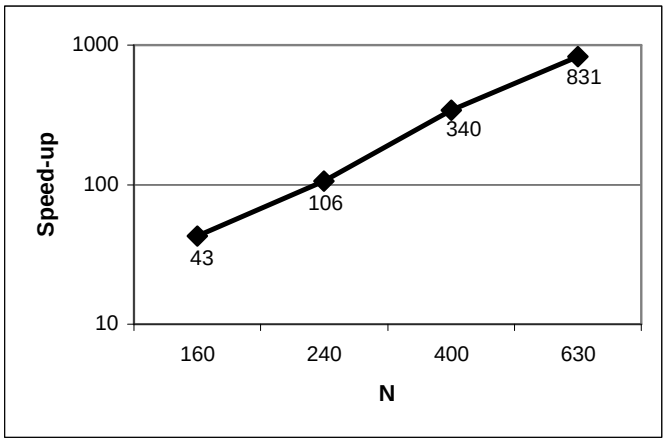


Figure 26. Speed-up between the original MTR algorithm and the accelerated version.

Table 1 contains the list of values which were obtained by the first experiment. It is obvious that both modifications of the MTR algorithm generate comparable values (number of triangles and vertices) only the computational time is significantly different.

GENUS 3	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 26 and in Table 1, represents a desired level of 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 get 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 has been $[<x_{min},x_{max}>,<y_{min},y_{max}>,<z_{min},z_{max}>]=[<-16,16>,<-16,16>,<-16,16>]$.

The histogram in Figure 27 illustrates that the triangular mesh generated by the accelerated version of the MTR method consists of triangles with the similar shape properties.

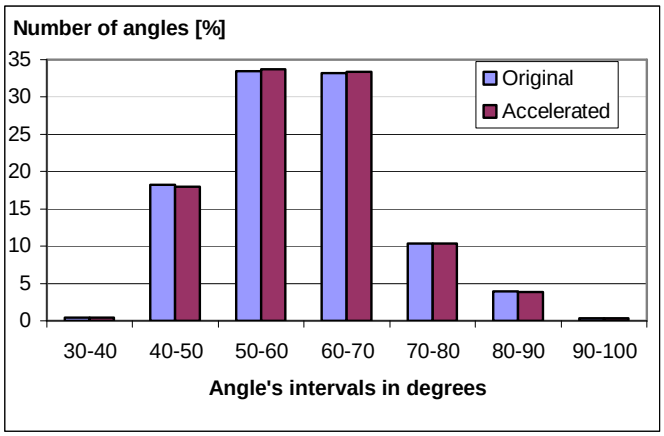


Figure 27. Histogram of the triangles' shape (angle distribution) for the MTR algorithms; generated for N=630, see Table 1.

3.1.5. Conclusion

The original Marching triangles algorithm has been significantly accelerated and its use is also suitable for triangulation of more detailed implicit objects now.

Note that the presented approach in this section has been published in [vii] of the author publications.

3.2. Edge spinning algorithm and its acceleration

In this section, the new algorithm for polygonization of the implicit surfaces will be introduced. The Edge spinning (ES) method put emphasis on the shape of triangles generated and on the polygonization speed as well. The algorithm is a variant of marching triangles methods [1], [19], [20], [21], i.e. it is based on the continuation (surface tracking) scheme.

3.2.1. 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 winged 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).

3.2.2. 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 such point. Several disjoint surfaces can be polygonized from a starting point for each of them. The whole algorithm consists of the following steps.

Algorithm 4. Edge spinning principle.

1. Find a starting point $\mathbf{p}_0$.
2. Create the first triangle T_0, see Figure 28.
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

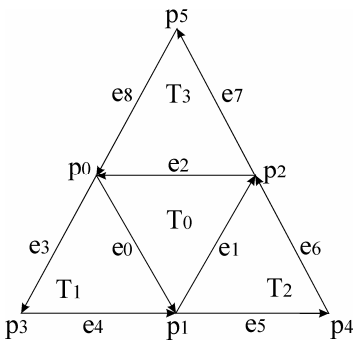


Figure 28. The first steps of the Edge spinning algorithm.

3.2.3. Starting point

There are several methods for finding a starting point on an implicit surface. These algorithms can be based on some random search method as in [5] or on more sophisticated approach. In [40], searching in constant direction from an interior of an implicit object is used.

In our approach, we use a simple algorithm for finding a starting point. A starting point is sought from any place in defined area in direction of a gradient vector ∇f of an implicit function *f*. The algorithm looks for a point $\mathbf{p}_0$ that satisfy the equation $f(\mathbf{p}_0) = 0$.

3.2.4. First triangle

The first triangle in polygonization is assumed to lie near a tangent plane of the starting point $\mathbf{p}_0$ that is on the implicit surface.

Algorithm 5. Creating of the first triangle.

1. Determine the normal vector $\mathbf{n} = (n_x,n_y,n_z)$ in the starting point $\mathbf{p}_0$, see Figure 29. $\mathbf{n} = \nabla\mathbf{f} / \|\nabla\mathbf{f}\|$
2. Determine the tangent vector **t** as in [19].
   If ($n_x > 0.5$) or ($n_y > 0.5$) then $\mathbf{t} = (n_y, -n_x, 0)$; else $\mathbf{t} = (-n_z, 0, n_x)$.
3. Use the tangent vector **t** as a fictive active edge and use the algorithm edge spinning (described bellow) for computation coordinates of the second point $\mathbf{p}_1$. The pair of points ($\mathbf{p}_0$, $\mathbf{p}_1$) forms the first edge e_0.
4. Polygonize the first edge e_0 with the edge spinning algorithm for getting the third point $\mathbf{p}_2$. Points ($\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$) and edges (e_0, e_1, e_2) form the first triangle T_0.

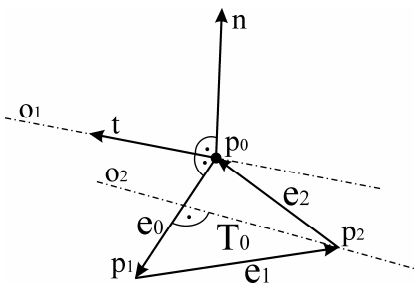


Figure 29. First triangle generation.

3.2.5. 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, [19]. Differential properties, [35], 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 (level of detail) 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 30) and axis *o* of the actual edge *e*; this guarantees that the new generated triangle is well shaped (isosceles at least).

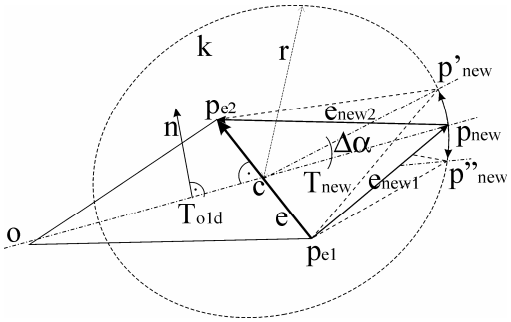


Figure 30. The root finding principle.

Then, the algorithm is as follows:

Algorithm 6. Root finding of the Edge spinning method.

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 30. Let the angle of the initial position be $\alpha=0$.
2. Compute the function 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 $-\Delta\alpha$.

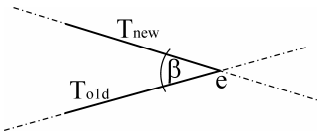


Figure 31. Angle between two triangles; the view is in direction of the edge's vector *e*.

4. Let the functional values be $f_1 = f(\alpha)$ and $f_2 = f(\alpha\pm\Delta\alpha)$; update the 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 31) then point $\mathbf{p}_{new}$ is accepted; else point $\mathbf{p}_{new}$ is rejected and return to step 4.

3.2.6. Active edge polygonization

Polygonization of an active edge *e* consists of several steps. At first, the algorithm checks adjacent active edges of the active edge *e* and determines which of following cases appeared, see Figure 32.

- If ($\alpha_i < \alpha_{lim\_1}$) then case a); i = 1, 2.
- If ($\alpha_2 < \alpha_{lim\_2}$) and ($||p_{e1} - p_{r\_e2}|| < \delta_{lim\_1}$) then case a); analogically for α_1.
- If ($\alpha_2 > \alpha_{lim\_3}$) and ($||p_{e1} - p_{r\_e2}|| < \delta_{lim\_2}$) then case b); analogically for α_1.
- else case c)

Note that the relations among limit angles are $\alpha_{lim\_1} < \alpha_{lim\_2} \leq \alpha_{lim\_3}$.

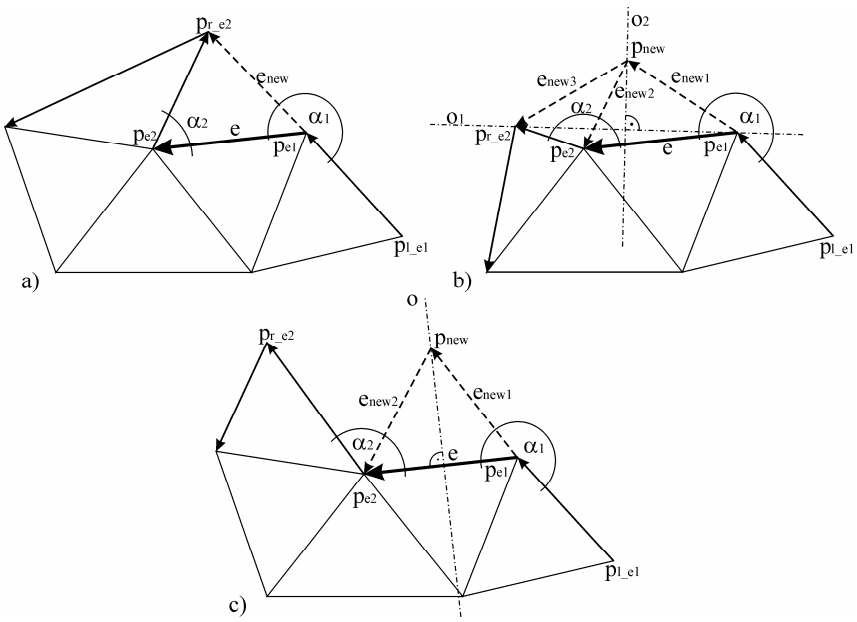


Figure 32. The possible cases for polygonization of an active edge.

Possible cases which are illustrated in Figure 32 are:

a) In this case, algorithm creates a new one triangle and includes a new active edge e_{new} to the end of the active edges list.
b) In some situations, the length of certain edges can be shorter then tolerable limit. In this case, algorithm must repair the length of the new edges e_{new1} and e_{new3} to achieve better shapes of next triangles. The axis o_1 (see Figure 32) is used as

a fictive active edge for the algorithm edge spinning and the new point $\mathbf{p}_{new}$ is created as well as two new triangles.

c) In all the other situations, the edge *e* is polygonized by the standard algorithm edge spinning.

3.2.7. Distance test

To preserve the correct topology and the shape of the mesh triangles it is necessary to perform the distance check between the new triangle and a border of already triangulated area. Therefore, each new generated point $\mathbf{p}_{new}$ must be checked for distance with all the other points which lie in active edges. Let the point $\mathbf{p}_{min}$ be the nearest point to this new point $\mathbf{p}_{new}$ and distance between both points is $\delta = ||\mathbf{p}_{new} - \mathbf{p}_{min}||$. Further, let $\mathbf{p}_{min}$ not lie in the active edges which are in the neighborhood of both active edges which contain the point $\mathbf{p}_{new}$. Then, there are two cases described in Figure 33.

a) If $\delta < \delta_{lim\_3}$ then the new point $\mathbf{p}_{new}$ is replaced with the point $\mathbf{p}_{min}$.

b) If $\delta < \delta_{lim\_4}$ then a new triangle must be created between the new point $\mathbf{p}_{new}$ and one of two active edges which contain the point $\mathbf{p}_{min}$. , i.e. either the triangle ($\mathbf{p}_{min}$, $\mathbf{p}_{new}$, $\mathbf{p}_{r\_min}$) or the triangle ($\mathbf{p}_{l\_min}$, $\mathbf{p}_{new}$, $\mathbf{p}_{min}$), see Figure 33b. The decision, which active edge will be used, depends on angles α_1, α_2. The angles α_i, i = 1, 2 are in interval $<0, \pi>$ and therefore, the triangle with the angle α_i that is better approximation of angle 90° is chosen.

Note that the relation between distance limits is $\delta_{lim\_3} < \delta_{lim\_4}$.

The situation described in Figure 33 a) and b) is similar for both cases now. Point $\mathbf{p}_{new}$ is contained in four active edges e_1, e_2, e_3, e_4 and a border of already triangulated area intersects itself on it. Solution of the problem will be introduced on case b) and solution for case a) is analogically. Let the four active edges be divided into pairs; the left pair is (e_3, e_2) and the right pair is (e_1, e_4). One of these pairs will be polygonized and the second one will be cached in memory for later use. The solution depends on angles β_1, β_2, see Figure 33b. If ($\beta_1 < \beta_2$) then the left pair (e_3, e_2) is polygonized; else the right pair (e_1, e_4) of active edges is polygonized. In both cases, the second pair that is not polygonized is deleted from the list of active edges and the point $\mathbf{p}_{new}$ is contained only in one pair of active edges.

In Figure 33b, the first case is valid ($\beta_1 < \beta_2$), i.e. the active edges (e_3, e_2) are polygonized in order that depends on angles γ_3, γ_2. If ($\gamma_3 < \gamma_2$) then the active edge e_3 is polygonized at first; else the active edge e_2 is polygonized first.

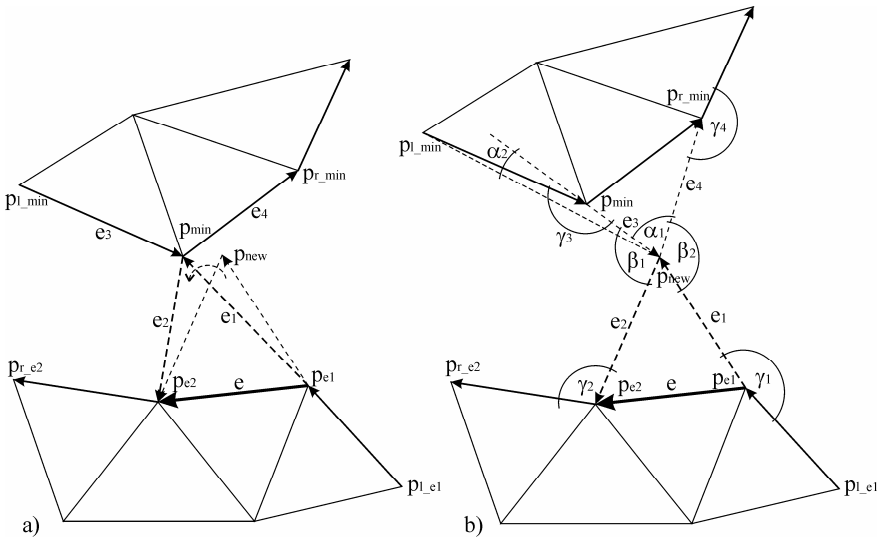


Figure 33. The possible cases for the distance test.

Now, the border of the triangulated area does not cross itself in the point $\mathbf{p}_{new}$ and the recently polygonized pair of edges is removed from the active edges list. The previously cached pair of edges must be returned into the list of active edges.

3.2.8. 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.

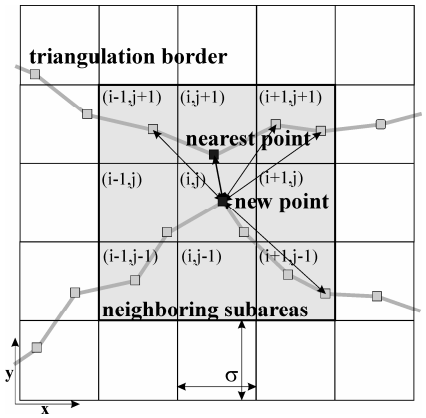


Figure 34. The space subdivision scheme for the Edge spinning algorithm.

Advantageous solution is dividing of space into sub-spaces (sub-areas), similarly as in case of the Marching triangles algorithm described 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 23.

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 34, and δ_{lim} is the limit distance for distance check.

The original algorithm checks distances with the algorithm complexity *O(N)*, where *N* is a number of points on the triangulation border. The accelerated distance test is of algorithm complexity O(*M*), where *M* is a number of points in adjacent sub-areas and M<<N. Figure 34 shows 9 possible sub-areas in E^2 case, there are 27 possible sub-areas in E^3 case, see Figure 35.

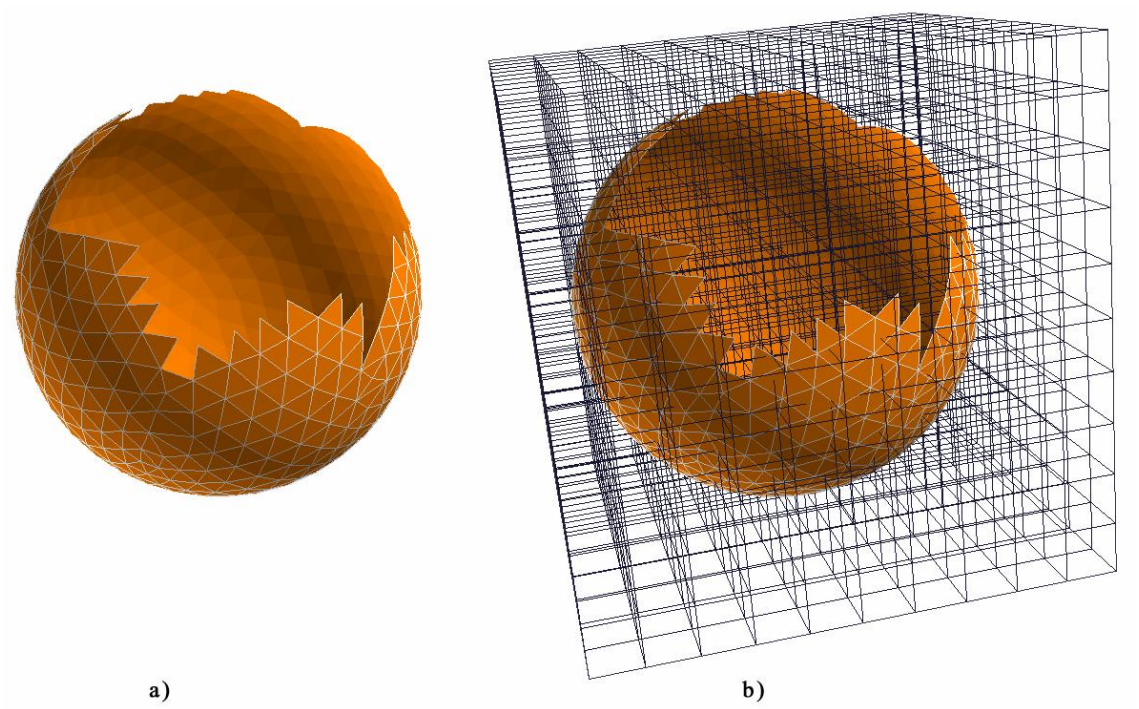


Figure 35. The algorithm has to perform the distance test among a) all points lying on the triangulation border; b) only points lying in the adjacent sub-areas to the new point.

3.2.9. Experimental results

Experimental results are divided into three parts where at first, the quality of triangles is compared among Edge spinning, Marching triangles and Marching cubes algorithms. Second experiment illustrates effectiveness of the space subdivision acceleration

technique and the last results have been achieved by comparison of Marching cubes and Edge spinning methods, including speed and quality, on various implicit objects.

Quality of triangular meshes

At first, we compare a quality of triangles generated by the Marching triangles [19], Edge spinning, and the Marching cubes [5] algorithms. For visual comparison, Figure 36 shows the Genus 3 object polygonized by these algorithms.

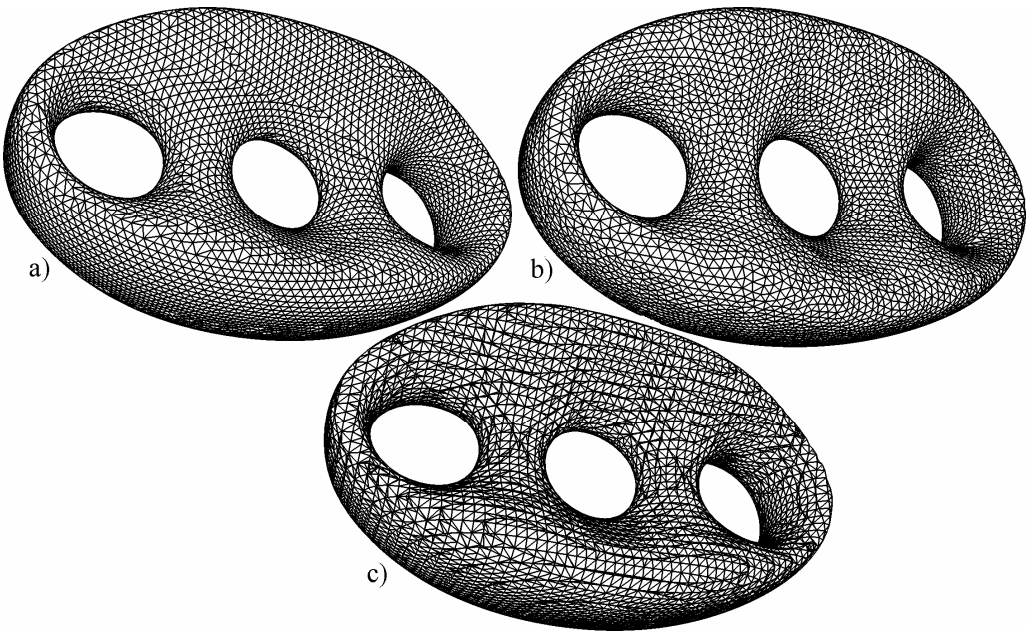


Figure 36. The Genus 3 object generated by a) the Edge spinning; b) the Marching triangles; c) the Marching cubes algorithm.

The percentage ratio of the angles incidence is shown in Figure 37. 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.

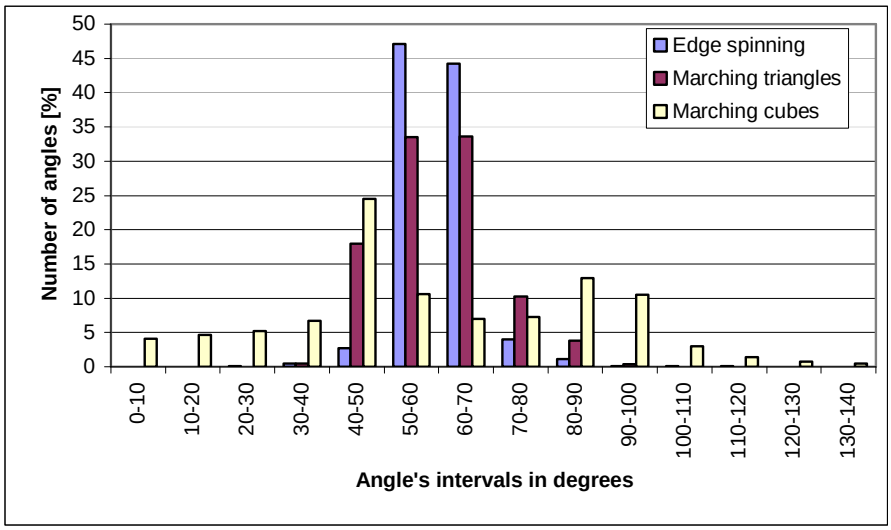


Figure 37. Histogram of angle distribution of triangular mesh, generated for Genus 3 object with average length of triangles edges set to 0.04 (it represents the desired level of detail - LOD).

Acceleration by the space subdivision

In the next experiment, we will have a look at speed comparison between the accelerated and the original ES algorithm. If we divide the computational time into two parts, the polygonization time and the distance check time, Figure 38 shows the percentage ratio of time with and without acceleration. It is obvious that the time, needed for distance checking, was significantly decreased.

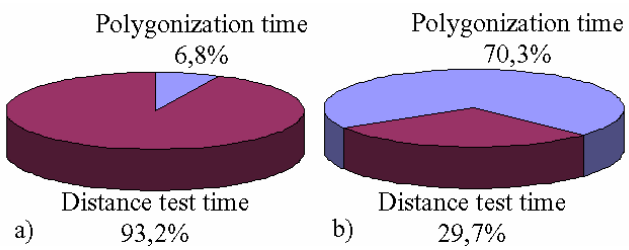


Figure 38. The time ratio between the Polygonization time and the Distance test time of the ES algorithm; a) without the space subdivision scheme, b) with the space subdivisions equal to 100.

Table 2 contains computing times measured with various space subdivisions as well as speed-up achieved. For better illustration, the values from the Table 2 are graphically shown in Figure 39.

subdivision	N/A	10	20	30	40	50	60	70	80	90	100
time [ms]	12609	7791	5969	4056	3024	2563	2284	2143	2033	1993	1943
speed-up	1,00	1,62	2,11	3,11	4,17	4,92	5,52	5,88	6,20	6,33	6,49

Table 2. Computational time and speed-up achieved in dependence on space subdivisions used. Genus 3 object, 331 414 triangles, 165 703 vertices, polygonization area $[<x_{min},x_{max}>, <y_{min},y_{max}>, <z_{min},z_{max}>] = [<-16,16>, <-16,16>, <-16,16>]$.

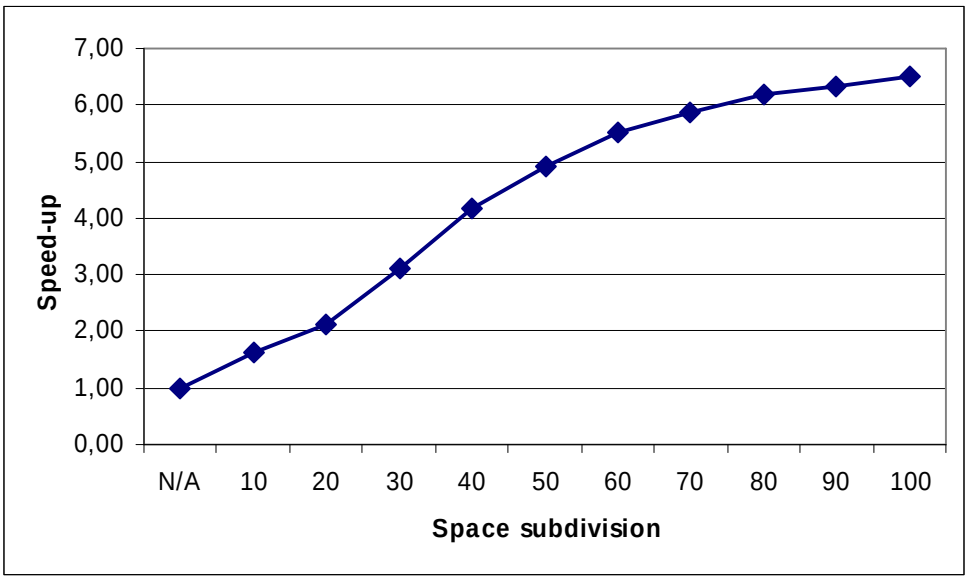


Figure 39. Comparison of speed-up depending on space subdivision used.

The results proved that this acceleration technique is effective and simple for implementation as well.

Comparison between the Edge spinning and the Marching cubes algorithms

Our next experiment is aimed at detailed comparison of Edge spinning and the Marching cubes algorithms. The measured values from the experiment are in Table 3. The space subdivisions for the Edge spinning algorithm has been equal to 100 and the polygonization area has been the same as in previous test. The values have been achieved with a variable lowest level of detail (LOD) because we want the number of generated triangles to be similar. Note that for the Marching cubes algorithm, the LOD value represents a size of cube cells. The experiment has been performed on implicit objects Genus, Jack, Morph and Spiral whose pictures are shown in Figure 40.

Table 3 contains the number of triangles and vertices generated. The value *Angle err* is proportional to surface curvature and means the average deviation between surface normal vectors at points sharing an edge. For the Edge spinning algorithm, it corresponds to a_{err} given at the beginning of the polygonization. The value *Centroid*

angle err represents the deviation between the normal vector of a triangle and the function normal vector computed at the centroid of the triangle. Note that the real normal vector is measured numerically from the implicit function at a given point.

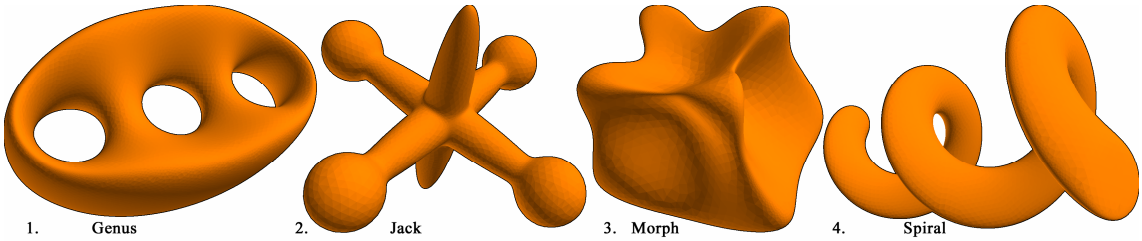


Figure 40. Implicit objects used in the experiment.

The values *Alg dist avg*, *Euc dist avg*, *Taub dist avg* measure the approximation quality as an average distance of a triangle from the real implicit surface. They are measured at a gravity centre of each triangle. The distance is either algebraic (*Alg dist avg*) or real Euclidian (*Euc dist avg*) or the Taubian [39] (*Taub dist avg*).

	Edge spinning				Marching cubes			
	Genus	**Jack**	**Morph**	**Spiral**	**Genus**	**Jack**	**Morph**	**Spiral**
LOD	0,04	0,02	0,03	0,04	0,05	0,02	0,03	0,05
Triangles	331 414	332 580	117 342	191442	334 816	327 208	134 552	201 908
Verices	165 703	166 290	58 671	95 723	167 404	163 606	67 274	100 954
Angle error	7,89E-03	1,02E-02	1,29E-02	1,32E-02	8,12E-03	1,06E-02	1,26E-02	1,40E-02
Centroid angle err.	1,84E-03	1,80E-03	2,73E-03	3,11E-03	2,29E-03	3,17E-03	3,87E-03	4,70E-03
Alg dist avg	0,16	2,50E-04	7,25E-04	6,68E-04	0,2	3,27E-04	8,32E-04	8,64E-04
Euc dist avg	1,08E-04	8,63E-05	1,64E-04	2,26E-04	1,41E-04	1,16E-04	1,90E-04	2,86E-04
Taub dist avg	1,08E-04	8,64E-05	1,64E-04	2,26E-04	1,41E-04	1,15E-04	1,90E-04	2,87E-04
Angle criterion	0,86	0,86	0,85	0,85	0,37	0,37	0,37	0,36
Edge length crit.	0,91	0,91	0,90	0,90	0,53	0,53	0,53	0,52
Time [ms]	1 892	3 345	941	2 243	1 692	2 053	831	1 943
Avg time [ms]	5,71	10,06	8,02	11,72	5,05	6,27	6,18	9,62
Time ratio ES/MC	1,13	1,60	1,30	1,22	---	---	---	---
Time ratio MC/ES	---	---	---	---	0,89	0,62	0,77	0,82

Table 3. Values from the experiment measured by the Edge spinning and the Marching cubes algorithms on various implicit objects.

Note that the algebraic distance (function value) strongly depends on the given implicit function and it is only proportional to the real distance. It is only useful for comparing

algorithms properties on the same objects. The Euclidian distance has been measured between a triangle centroid and its corresponding surface point; note that it is computed numerically, see Algorithm 9 on page 53 for details. For the given implicit functions, it can be seen that the Taubian distance is a good approximation of the real distance.

The value *Angle criterion* means the criterion of the ratio of the smallest angle to the largest angle of a triangle and the value *Edge length criterion* means the criterion of the ratio of the shortest edge to the longest edge of a triangle. These values show the quality of resulting triangles generated.

The value *Time* shows the measured computational time of each algorithm and the value *Time avg* represents an average time needed for creating of one thousand triangles. *Time ratio* values represent a speed comparison between the both algorithms.

Note that all the criteria mentioned above have been defined in section 2.1 and will be used in the same meaning in following section as well.

3.2.10. Conclusion

In this section, the new principle for polygonization of implicit surfaces has been presented. The algorithm marches over the object's surface and computes the accurate coordinates of new points by spinning the edges of already generated triangles. The Edge spinning algorithm generates triangular meshes of excellent quality and the polygonization speed is, with using the space subdivision scheme, comparable with the well-known Marching cubes algorithm. The space subdivision scheme seems to be an effective way for speed-up of such type of geometric algorithms.

Usage of the Edge spinning algorithm is limited for implicit surfaces which comply C^1 continuity which is a common problem of surface tracking approaches.

Note that the presented approaches of this section have been published in [iv], [viii] and [ix] of the author publications.

3.3. Adaptive Edge spinning algorithm

Many polygonization algorithms adaptively or non-adaptively create polygonal meshes without a proper definition of an approximation error that is requested in result. Usually, the algorithms allow to user to set a level of detail (min/max size of triangles, number of divisions in axes, etc.) which only has a little relation to the resulting approximation quality. The quality strongly depends on ratio between size of implicit objects and size of triangles, size of computational area, etc.

Our algorithm defines the approximation error that is proportional to surface curvature estimation, see variable a_{err} in section 3.3.1. The desired error is given at the beginning of the computation and it is preserved for all triangles during the whole polygonization. The resulting polygonal mesh consists of well shaped and adaptively sized triangles and moreover, it preserved the given approximation quality as well.

In this section, the adaptive extension of the Edge spinning algorithm will be presented. The method approximates an implicit surface by a triangular mesh according to local estimation of surface curvature. The polygonal mesh is created with respect to preserve given approximation error as well as to achieve the best possible shape of triangles generated.

3.3.1. Principle of the algorithm

The algorithm is based on the surface tracking scheme as methods mentioned in previous sections. Its principle and basic steps are analogical to the original non-adaptive Edge spinning method.

Because the adaptive approach operates with triangles of different size, some steps inside are different from the original method and will be described in following sections.

The whole algorithm consists of the following steps.

Algorithm 7. Adaptive Edge spinning principle.

1. Initialize the polygonization:
   a) Find the starting point p_0 and create the first triangle T_0.
   b) Include the edges (e_0,e_1,e_2,) of the first triangle T_0 into the active edges list.
2. Polygonize the first active edge *e* from the active edges list.
3. Update the AEL; delete the currently polygonized active edge *e* and include the new generated active edge/s at the end of the list.
4. If the active edges list is not empty return to step 2.

Note that at the beginning of polygonization, there are two variables important for computation:

- LOD_{max} – the maximal length of triangles' edges, i.e. maximal level of detail;
- a_{err} – desired accuracy of approximation, i.e. desired maximal angle between normal vectors at points lying on the same edge of a triangle; this variable represent a measure of dependence on surface curvature.

The whole polygonization is controlled by these criteria and new triangles generated are created adaptively to preserve the accuracy.

3.3.2. Root finding with curvature estimation

New generated points are sought on a circle as in the original algorithm but the finding circle radius is proportional to the estimated surface curvature now.

The surface curvature radius r_c between points $\mathbf{p}_1$, $\mathbf{p}_2$ with their normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$, see Figure 41, is estimated by the simple formula:

$$r_c = \frac{d}{a}, \tag{43}$$

where d is the distance between the points $\mathbf{p}_1$, $\mathbf{p}_2$ and a is the angle between the surface normals $\mathbf{n}_1$, $\mathbf{n}_2$.

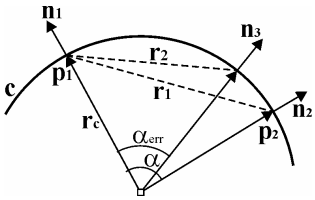


Figure 41. The circle c with radius r_c of surface curvature between points $\mathbf{p}_1$, $\mathbf{p}_2$ and estimation of radius r_2 of finding circle according to desired approximation error a_{err}.

The new radius r_2 of the finding circle is then computed as follows.

$$r_2 = k \cdot r_c \cdot \sqrt{2 \cdot (1 - \cos a_{err})}, \tag{44}$$

where k is a constant, r_c is the estimated radius of surface curvature and a_{err} is the required approximation error given at the beginning of the polygonization process. Because it is an estimation, we used the k = 0.8 constant just to be surer that the new triangle will satisfy the desired accuracy.

Note that this formula has been derived from the second cosine theorem $c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos a$ with the presumption a = b = r_c, see Figure 41. The initial radius r_1 of the circle c_1 is proportional to the length of the current active edge e to a new triangle T_{new} be equilateral.

Limitations of the final radius:

if ($r_2 < r_{min}$) then $r_2 = r_{min}$,

if ($r_2 > r_{max}$) then $r_2 = r_{max}$,

where $r_{min} = \frac{1}{10} r_{max}$ and r_{max} is a limit value derived from the maximal level of detail LOD_{max} which edges of the new triangle have to satisfy.

For creating of a new triangle, the radius of surface curvature r_c is evaluated three times among pairs of points ($\mathbf{p}_1$, $\mathbf{p}_{init}$), ($\mathbf{p}_2$, $\mathbf{p}_{init}$), ($\mathbf{p}_s$, $\mathbf{p}_{init}$), where $\mathbf{p}_1$, $\mathbf{p}_2$ are points of the current active edge e, $\mathbf{p}_s$ is its midpoint and $\mathbf{p}_{init}$ is the point of intersection of the circle c_1 with the plane defined by the triangle T_{old}, see Figure 42. The final r_c is taken as minimum from these three.

Determination of the point $\mathbf{p}_{new}$ location is then analogical to Algorithm 6.

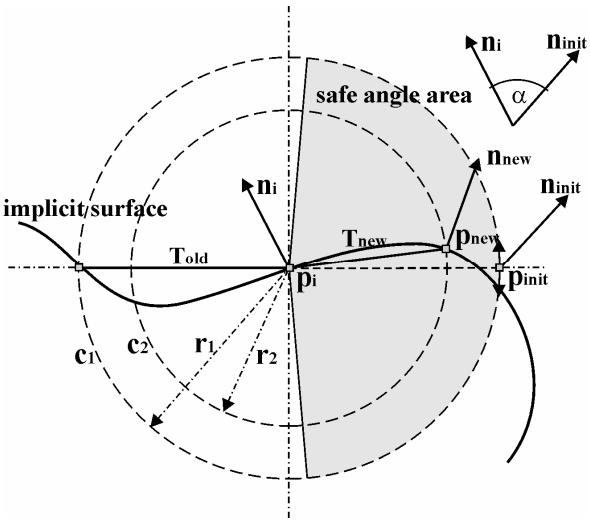


Figure 42. The finding circle radius estimation.

3.3.3. Root finding on a sharp edge

Let us assume that the standard edge spinning root finding algorithm presented above has found the point $\mathbf{p}_{new}$. The algorithm then determines the surface normal vector $\mathbf{n}_{new}$ at this point and computes the angle α between normal vectors $\mathbf{n}_{new}$ and $\mathbf{n}_s$. The vector $\mathbf{n}_s$ is measured at mid-point **s** of the active edge *e*, see Figure 43. If the angle α is greater then some user-specified threshold α_{lim_edge} (limit edge angle) then the algorithm will look for a new edge point as follows.

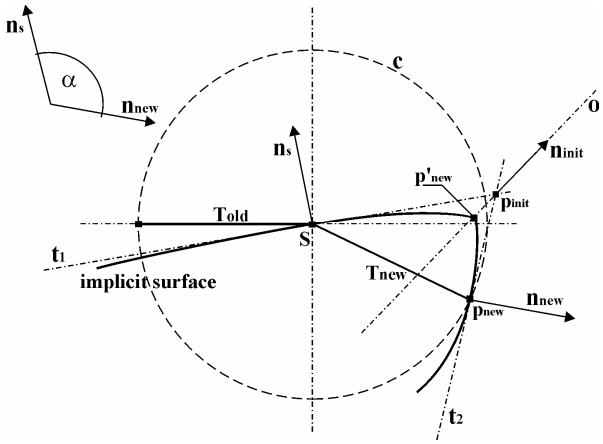


Figure 43. The principle of root finding algorithm for sharp edges.

Algorithm 8. Edge detection for the Edge spinning method.

1. Compute coordinates of the point $\mathbf{p}_{init}$ as an intersection of the three planes, tangent planes $\mathbf{t}_1$ and $\mathbf{t}_2$, and the plane in which the seeking circle *c* lies, see Figure 43.

2. Apply the straight root finding algorithm described in section 3.3.4 and find the new point $\mathbf{p}'_{new}$.

Note that the algorithm needs an accurate determination of surface normal vectors, i.e. accurate computation of a function gradient. Therefore, implicit objects should be modeled by F-Rep, [34], because objects defined by min/max operations are not good differentiable, [27], [35].

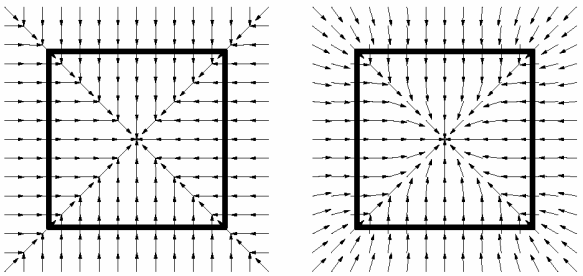


Figure 44. A square modeled as intersection of four half-spaces; left: by min/max operations; right: by the F-Rep operations; taken from [27].

The gradient array of a *square*, modeled by min/max and F-Rep operations, is illustrated in Figure 44. The picture shows that the min/max operations create objects with poor differential properties.

3.3.4. Straight root finding algorithm

The algorithm starts from an initial point $\mathbf{p}_{init}$ (see Figure 45) and supposes that the implicit surface is at least C^0 continuity.

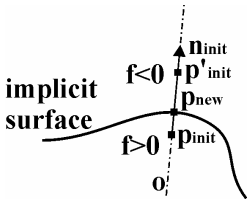


Figure 45. Principle of root-finding in straight direction.

The algorithm continues as follows.

Algorithm 9. Straight root finding algorithm.

1. At point $\mathbf{p}_{init}$, compute the surface normal vector $\mathbf{n}_{init}$ that defines the seeking axis *o*.
2. Compute coordinates of point $\mathbf{p}'_{init}$ with distance δ from point $\mathbf{p}_{init}$ in direction $\mathbf{n}_{init} * \text{sign}(f(\mathbf{p}_{init}))$; where δ is the step length.
3. Determine function values *f, f'* at points $\mathbf{p}_{init}$, $\mathbf{p}'_{init}$.

4. Check next two cases.
   a) If these points lie on opposite sides of implicit surface, i.e. $(f*f') < 0$; compute the exact coordinates of the point $\mathbf{p}_{new}$ by binary subdivision between these points.
   b) If the points $\mathbf{p}_{init}$, $\mathbf{p}'_{init}$ lie on the same side of the surface then $\mathbf{p}_{init} = \mathbf{p}'_{init}$ and return to step 2.

3.3.5. Polygonization of an active edge

Polygonization of an active edge *e* consists of several following steps.

Algorithm 10. Active edge polygonization.

1. Use the Edge spinning algorithm to find a new point $\mathbf{p}_{new}$ in front of the edge *e*.
2. Determine angles a_1, a_2 in front of points $\mathbf{p}_1$, $\mathbf{p}_2$ of the current edge *e*, see Figure 46.
3. Perform neighborhood test.
4. Perform distance test.

Neighborhood test

If the point $\mathbf{p}_{new}$ has been found, there are two cases illustrated in Figure 46. Decision between cases a) and b) depends on relation among angles α_1, α_2, α_n, see Figure 46. Let the angle α be $min(\alpha_1,\alpha_2)$. If $(\alpha < \alpha_{shape})$ then case a) else case b), see Figure 46. The limit shape angle is determined as $\alpha_{shape} = \alpha_n + \pi/6$, so the space for next triangles should be at least $\pi/6$; this constant just affect a shape of next generated triangles.

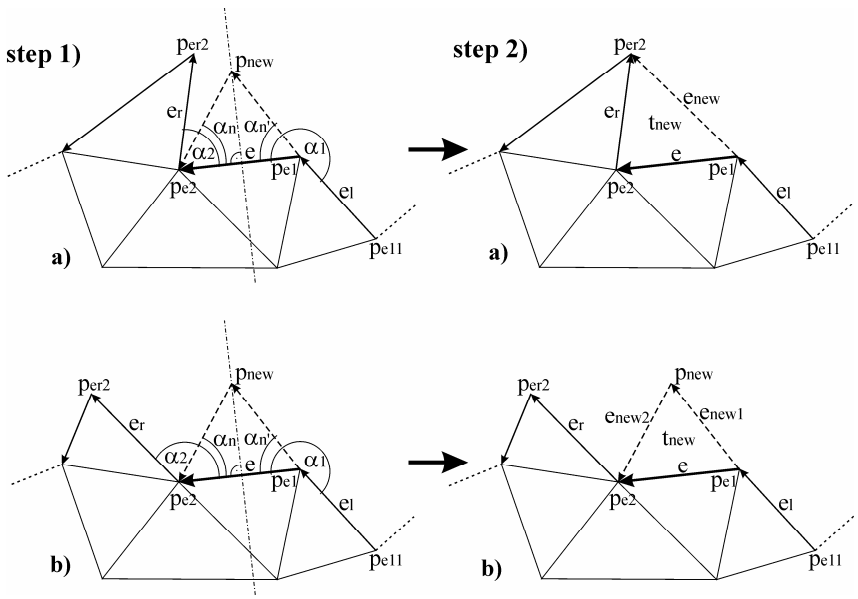


Figure 46. Polygonization of the active edge e.

If the point $\mathbf{p}_{new}$ is not found, angle α_n is not defined and the limit angle α_{shape} should be just less then π; we have chosen $\alpha_{shape} = 2/3 * \pi$.

a) In this case, a new triangle t_{new} is created by connecting the edge *e* with one of its neighbors, see step 2a.

b) The new triangle t_{new} is created by joining the active edge *e* and the new point $\mathbf{p}_{new}$, see step 2b.

In both cases, a bounding sphere is determined for the new triangle t_{new}. The bounding sphere is the minimal sphere that contains all three points of the triangle, i.e. the centre of the sphere lies in the plane defined by these three points.

Note if there is not a new triangle (the point $\mathbf{p}_{new}$ does not exist and case a) has not appeared) the bounding sphere of the active edge *e* is used. The next procedure is analogical for all cases.

Distance test

To preserve the correct topology, it is necessary to check each new generated triangle if it does not cross a surface already generated. It is sufficient to perform this test between the new triangle and a border of already triangulated area (i.e. active edges in *AEL*).

The algorithm will make the *nearest active edges list* (*NAEL*) to the new triangle t_{new}. Each active edge which is not adjacent to the current active edge *e* and which crosses the bounding sphere of the new triangle (or the edge *e*), is included into the list, see Figure 48, step 2. The extended bounding sphere is used for the new triangle created by the new point $\mathbf{p}_{new}$ (case b) because the algorithm should detect a collision in order to preserve well-shaped triangles. The new radius of the bounding sphere is computed as $r_2 = c*r_1$ and we used the constant $c = 1.3$.

If the *NAEL* list is empty then the new triangle t_{new} is finally created and the active edges list is updated.

In case a), Figure 46 step 2, the current active edge *e* and its neighbor edge e_r are deleted from the list and one new edge e_{new} is added at the end of the list. The new edge should be tested if it satisfies the condition of the surface curvature. If it does not then the new triangle will be split along the edge e_{new}, see section bellow.

In case b), Figure 46 step 2, the current active edge *e* is deleted from the list and two new edges e_{new1}, e_{new2} are added at the end of the list.

Note that if there is no new triangle to be created (the point $\mathbf{p}_{new}$ does not exist and case a) in Figure 46 has not appeared) the current active edge *e* is moved at the end of the *AEL* list and the whole Algorithm 7 will return back to step 2.

If the *NAEL* list is not empty then the situation has to be solved. The point $\mathbf{p}_{min}$ with the minimal distance from the current edge *e* is chosen from the *NAEL* list, see Figure 48, step 3.

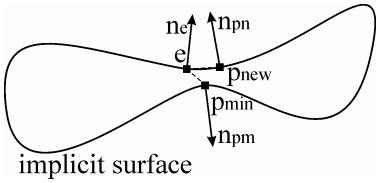


Figure 47. A problem of thin implicit objects.

This point has to satisfy a condition of thin objects as well. The current active edge *e* and the point $\mathbf{p}_{min}$ should not lie on the opposite sides of the implicit surface. Figure 47 illustrates the wrong situation.

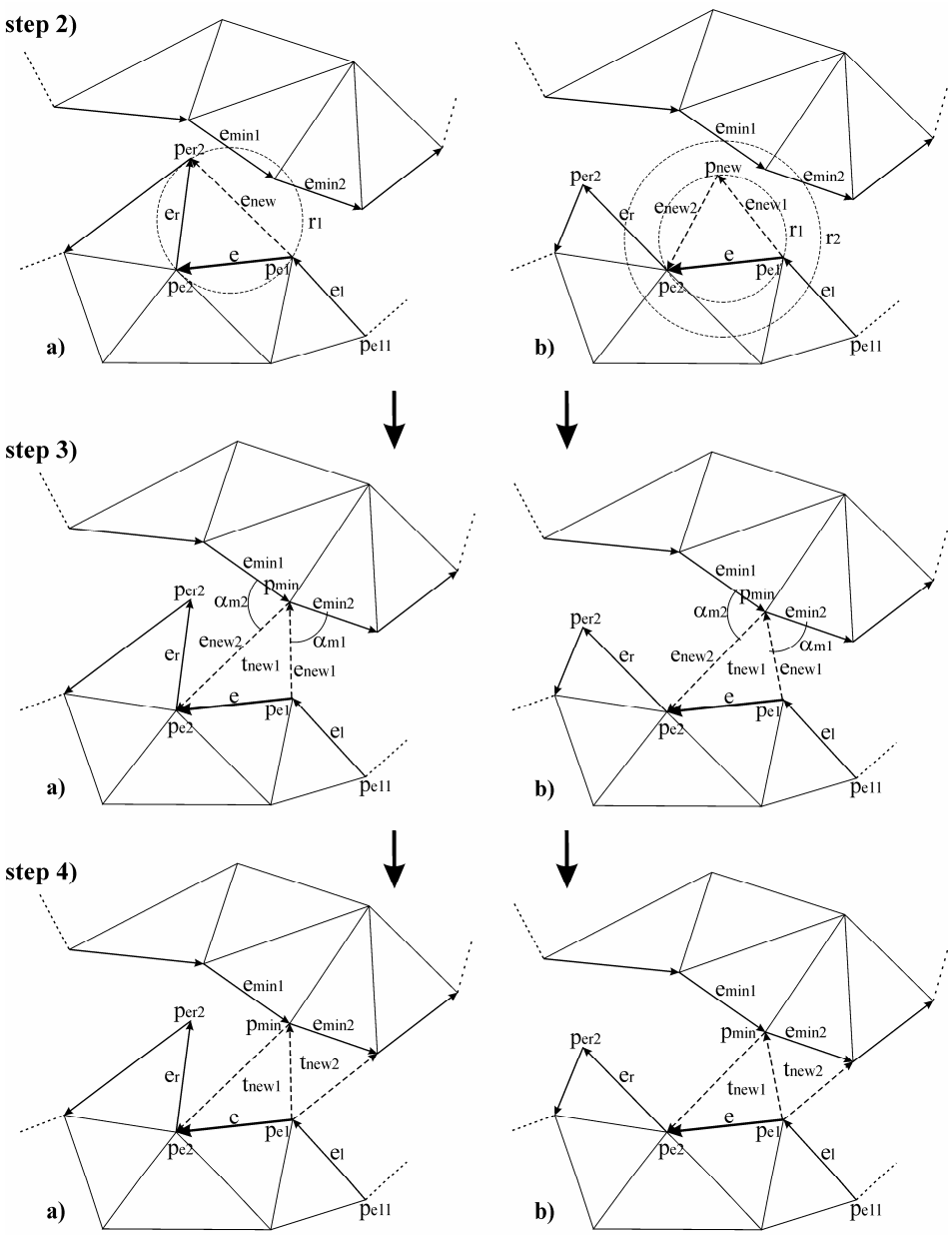


Figure 48. Solving of distance test.

If the correct point $\mathbf{p}_{min}$ is found, the new triangle t_{new} has to be changed and will be formed by the edge *e* and the point $\mathbf{p}_{min}$, i.e. by points ($\mathbf{p}_{e1}$,$\mathbf{p}_{min}$,$\mathbf{p}_{e2}$); the situation is described in Figure 48, step 3. The point $\mathbf{p}_{min}$ is owned by four active edges e_{new1}, e_{new2}, e_{min1}, e_{min2} and the border of already triangulated area intersects itself on it. This is not correct because each point that lies on the triangulation border should has only two neighborhood edges (left and right).

Solution of the problem is to triangulate two of four edges first. Let the four active edges be divided into pairs; the left pair be (e_{min1}, e_{new2}) and the right pair be (e_{new1}, e_{min2}). One of these pairs will be polygonized and the second one will be cached in memory for later use. The solution depends on angles α_{m1}, α_{m2}, see Figure 48, step 3. If ($\alpha_{m1} < \alpha_{m2}$) then the left pair is polygonized; else the right pair is polygonized.

In both cases, the recently polygonized pair is automatically removed from the list and the previously cached pair of edges is returned into the list. The point $\mathbf{p}_{min}$ is contained only in one pair of active edges and the border of the triangulated area is correct, see Figure 48, step 4.

Note that the polygonization of one pair of edges consists just of joining its end points by the edge and this second new triangle has to fulfill the empty *NAEL* list as well; otherwise the current active edge *e* is moved to the end of *AEL* list.

3.3.6. Splitting the new triangle

This process is evaluated only in cases when the new triangle has been created by connecting of two adjacent edges, i.e. situation illustrated in Figure 46, step 2a. If the new edge does not comply a condition of surface curvature the new triangle should be split. That means, see Figure 49; if the angle α between surface normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$ at points $\mathbf{p}_{e1}$, $\mathbf{p}_{er2}$ is greater than some limit α_{split_lim} then the new triangle will be split into two new triangles, see Figure 49, step 2.

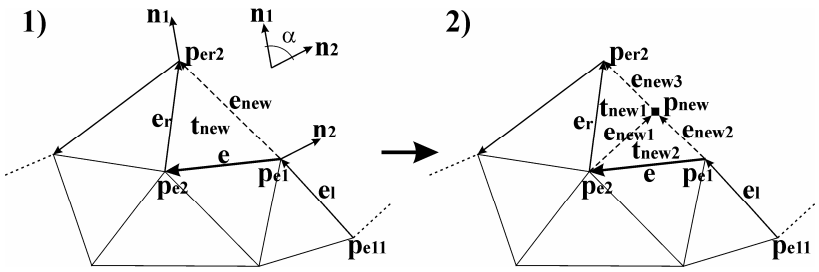


Figure 49. Splitting of the new triangle.

The point $\mathbf{p}_{new}$ is a midpoint of edge e_{new} and it does not lie on the implicit surface. Its correct coordinates are additionally computed by the straight root finding algorithm described in section 3.3.4.

3.3.7. Experimental results

Our first experiment is aimed at comparing dependence of approximation quality on variable *Angle error* a_{err} (an input variable, given at the beginning of polygonization, see Algorithm 7). The Adaptive Edge spinning algorithm change size of triangles generated according to it and resulting approximation quality strongly depends on it as well, see Table 4.

GENUS 3	Angle error set a_{err}		
	0,08	**0,04**	**0,02**
Triangles	32 246	75 650	182 502
Verices	16 119	37 821	91 247
Angle error	3,17E-02	2,30E-02	1,47E-02
Centroid angle error	7,60E-03	5,26E-03	3,29E-03
Alg dist avg	1,66	0,69	0,28
Euc dist avg	1,10E-03	4,52E-04	1,81E-04
Taub dist avg	1,10E-03	4,53E-04	1,82E-04
Angle criterion	0,72	0,74	0,75
Edge length criterion	0,82	0,83	0,84
Time [ms]	671	2 013	4 276

Table 4. Values of the object Genus 3 with the variable angle error set a_{err}.

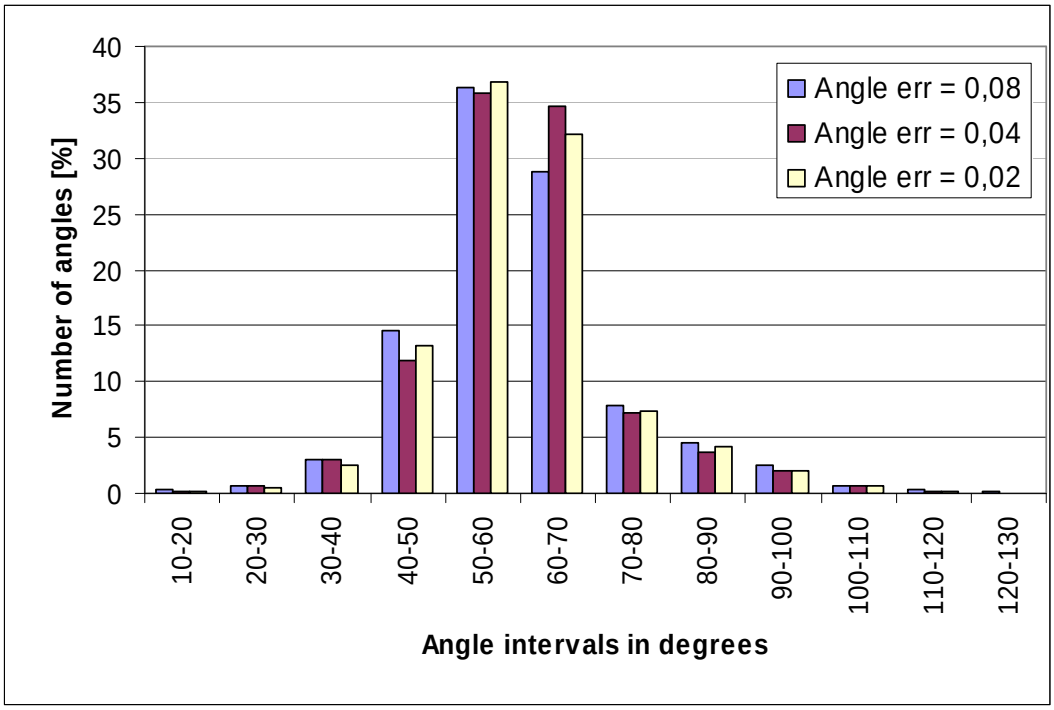


Figure 50. The histogram of triangles shape quality in dependence on angle error value.

The histogram in Figure 50 shows that the shape of triangles is not much influenced by the angle error variable and the Adaptive Edge spinning algorithm generates about 70% of triangles with angles in interval <50,70> degrees.

An another test is aimed at comparison of a surface approximation quality with the same starting LOD values for all polygonization methods, Adaptive edge spinning, Marching triangles and Marching cubes. The test is performed on the Jack object, introduced in [5] and it shows advantages of the adaptive approach.

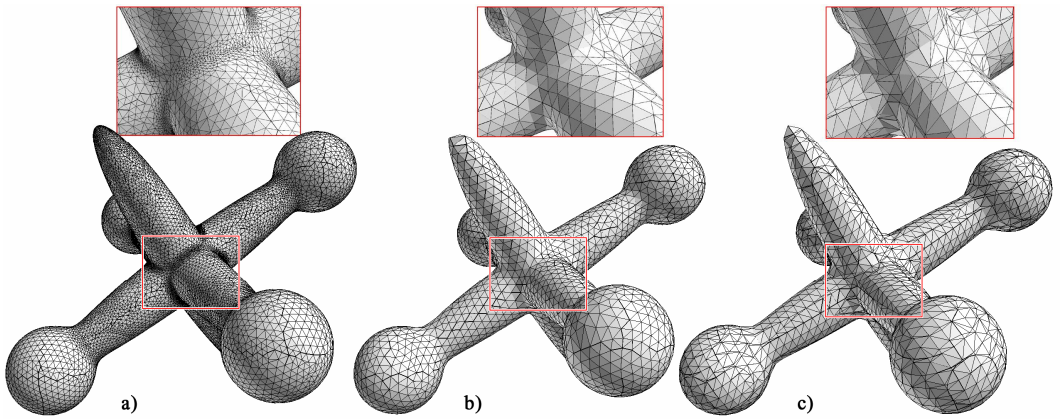


Figure 51. The Jack object generated by the a) Edge spinning, b) Marching triangles and c) Marching cubes algorithms. Details are zoomed for better illustration.

The adaptive Edge spinning algorithm shrinks the size of triangles in regions of higher curvature and therefore, the number of triangles is greater than that of generated by the other non-adaptive algorithms. The precision of polygonization is higher by about one order of magnitude, see Table 5.

JACK	Edge spinning	Marching triangles	Marching cubes
LOD	0,16	0,16	0,16
Triangles	34 256	6 107	6 552
Vertices	17 130	3 055	3 278
Angle err	3,33E-02	7,47E-02	7,54E-02
Centroid angle err	7,10E-03	1,40E-02	2,13E-02
Alg dist avg	2,39E-03	1,32E-02	1,67E-02
Euc dist avg	8,29E-04	4,62E-03	5,93E-03
Taub dist avg	8,30E-04	4,66E-03	5,97E-03
Angle criterion	0,700	0,729	0,377
Edge length criterion	0,806	0,828	0,536
Time [ms]	1 442	70	71
Time avg [ms]	42,09	11,46	10,83

Table 5. Values of the Jack object measured with the constant level of detail for all methods. Note that the Adaptive edge spinning algorithm has had the angle error value (a_{err}) set to 0,04

Figure 52a shows the object generated by the adaptive algorithm, so the number of triangles generated is higher in dependence on the surface curvature. In case of non-adaptive approaches, some parts of an object could be lost because the algorithm just connects thinner parts by large triangles depending on a given lowest level of detail; an example is shown in Figure 52b. The resulting image generated by the Marching cubes algorithm is shown in Figure 52c.

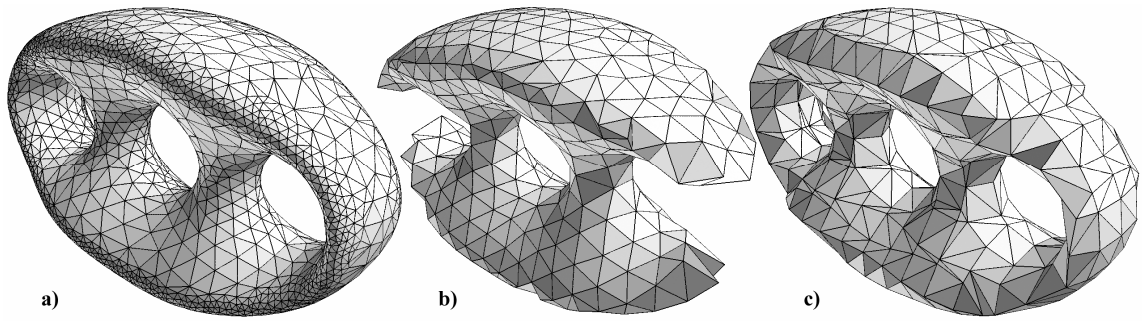


Figure 52. The Genus 3 object generated by the a) Adaptive edge Spinning algorithm; b) Marching triangles algorithm; and c) Marching cubes algorithm.

GENUS 3	Adaptive Edge spinning	Non-adaptive Edge spinning	Marching triangles	Marching cubes
LOD	0,16	0,016	0,016	0,016
Triangles	75 650	2 096 678	2 455 489	2 735 836
Verices	37 821	1 048 335	1 227 740	1 367 914
Angle error	2,30E-02	3,16E-03	2,92E-03	2,84E-03
Centroid angle error	5,26E-03	7,26E-04	6,88E-04	8,91E-04
Alg dist avg	0,69	2,53E-02	2,19E-02	2,49E-02
Euc dist avg	4,52E-04	1,70E-05	1,47E-05	1,72E-05
Taub dist avg	4,53E-04	1,70E-05	1,47E-05	1,72E-05
Angle criterion	0,74	0,90	0,72	0,37
Edge length criterion	0,83	0,94	0,82	0,53
Time [ms]	1 993	20 179	67 737	20 549

Table 6. Genus object polygonized by given algorithms with the same minimal level of detail. Note that the Adaptive edge spinning algorithm has had the angle error value (a_{err}) set to 0,04.

There is an opposite point of view to adaptive approaches and our next experiment is aimed at it. Table 6 contains values measured on Genus 3 object by the Adaptive edge spinning, non-adaptive Edge spinning and Marching triangles algorithms. The non-adaptive approaches have had the level of detail set to that minimal possible for the

adaptive method, i.e. they generate triangles of minimal size possible for the adaptive approach. In such case, the adaptive method generates as many triangles needed to achieve a desired accuracy and it minimizes a computational time as well as a number of triangles generated. To the contrary, the non-adaptive approaches generate unnecessarily huge number of triangles in flat regions and therefore, the accuracy is also unnecessarily high as well as the computational time is expensive.

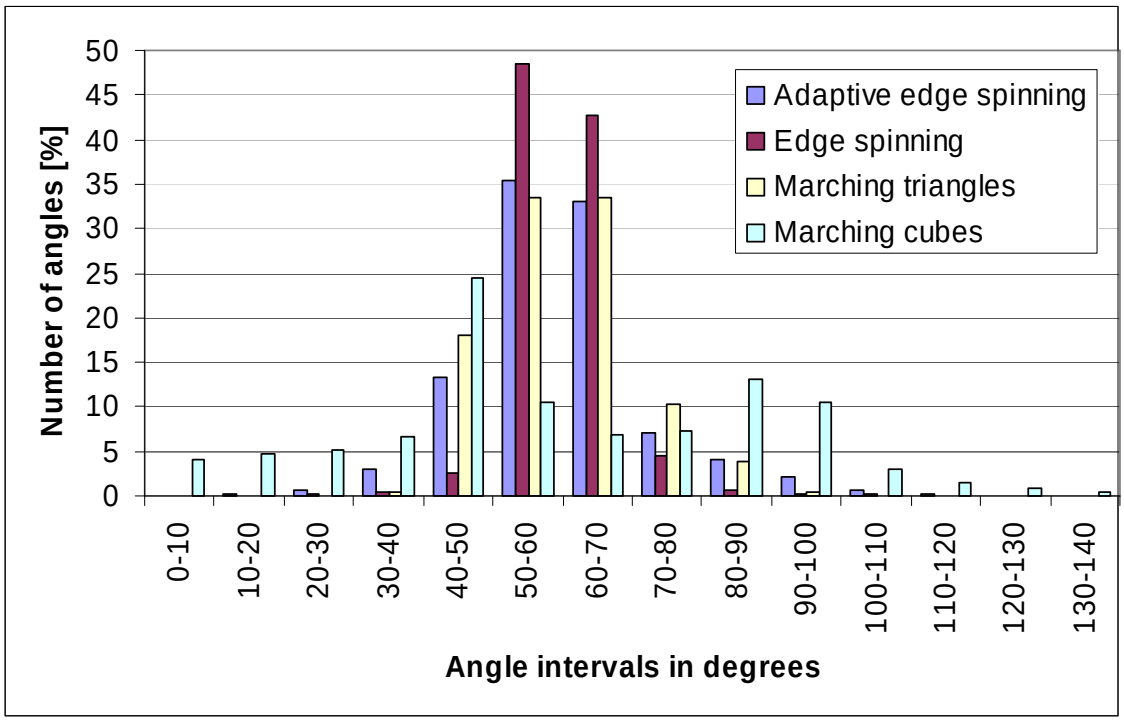


Figure 53. Histogram of angle distribution according to values in Table 6.

The histogram in Figure 53 shows angle distribution of triangular meshes generated by the given algorithms. In case of triangles shape quality, non-adaptive approaches have an advantage because these generate triangles of nearly constant size; therefore it is easier to achieve equilateral triangles. It is obvious that the classical Edge spinning algorithm generates triangular meshes of excellent quality but the adaptive approach is also very good although it musts operate with triangles of different size. The both edge spinning methods and the Marching triangles algorithm generate the most number of triangles with angles in interval <50, 70> degrees. The poorest triangular mesh is generated by the Marching cubes method.

Next experiment shows polygonization of simple implicit functions with sharp features. Figure 54 shows an object modeled as intersection of two spheres. The implicit object complies only the C^0 continuity and it is correctly polygonized by the proposed Adaptive edge spinning method. The picture a) is polygonized without the edge detection, i.e. the limit edge angle α_{lim_edge} is equal to π and the picture b) is polygonized with limit edge angle equal to $\pi/4$, see section 3.3.3 for details.

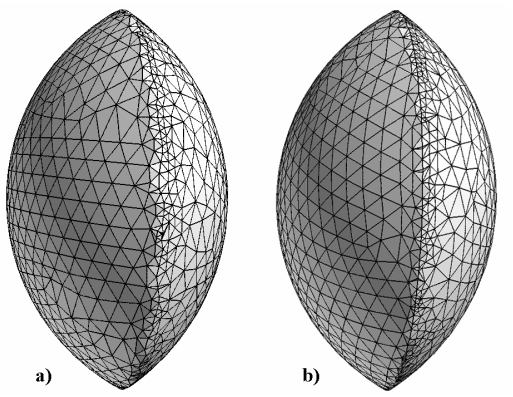


Figure 54. Intersection of two spheres generated by the Adaptive Edge spinning algorithm; with and without edge detection.

3.3.8. Conclusion

The new adaptive approach for polygonization of implicit surfaces has been presented in this section. The algorithm marches over the object's surface and computes the accurate coordinates of new points by spinning the edges of already generated triangles. Size of new triangles generated depends on the surface curvature estimation. We used the estimation by deviation of angles of adjacent points because it is simple and fast for computation. Our experiments proved its functionality as well. Other estimation techniques can be found in [3], [23], [42].

The algorithm can polygonize implicit surfaces which comply C^1 continuity, thin objects and some non-complex objects of C^0 continuity (an object should have only sharp edges, no sharp corners or more complex shapes).

The main advantage of our algorithm, in comparison to other methods, [1], [23], is its controlling of approximation quality during computation. The whole process is directed to achieve the desired accuracy given at the beginning and the algorithm maintains this requirement in all places of an implicit object (high/low curvature). It means that the resulting polygonal mesh does not consists only of well-shaped triangles, but moreover, the mesh satisfies predefined requirements of accuracy as well.

The presented approaches in this section have been published in [i], [ii], [iii] of the author publications.

3.4. Solving of Sharp features

Our previously developed methods have been able to polygonize implicit surfaces which comply C^1 continuity or have only simple sharp edges; no corners or more complicated shapes. A computation of a gradient vector (normal vectors) in areas of sharp features is influenced by a major error and surface approaches become unstable in such regions, see Figure 55.

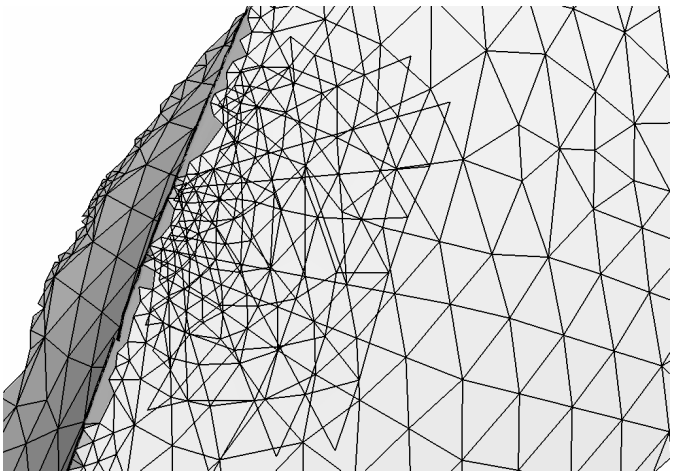


Figure 55. An edge of the Yutaka model. The Edge spinning algorithm become unstable in the region of the sharp edge and started to generate the already polygonized surface again.

In order to solve problem of sharp features we used a simple technique how to bypass it. Our technique supposes that an object is modeled by the F-Rep [22], which gives to it good differential properties, see Figure 44. It allows us to assume that an implicit surface has sharp edges only in its zero-set, i.e. at points $\mathbf{x}_i$ that satisfy equation $f(\mathbf{x}_i) = 0$. If the algorithm will look for some iso-value *e*, the equation will change to $f(\mathbf{x}_i) = \varepsilon$ and the implicit surface is then C^1 continuous, see Figure 56.

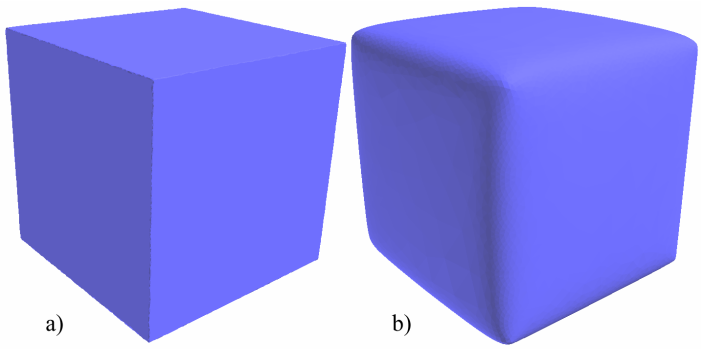


Figure 56. A cube modeled by the F-Rep as intersection of six half-spaces and polygonized with a) $\varepsilon = 0$ – classical approach, surface with sharp features and b) $\varepsilon = 0.1$, smooth surface.

The discussed implicit surface property allows us to construct an initial mesh that satisfies a desired accuracy according to surface curvature and consists of well-shaped triangles as well.

When the initial mesh is created, the algorithm has to find new positions of surface vertices $\mathbf{x}_i$ on the original implicit surface, i.e. $\varepsilon = 0$, $f(\mathbf{x}_i) = 0$. There are vary algorithms

that work with an initial mesh and iteratively adapt it to get more precise approximation, see section 2.7 for more details. In order to verify our approach, we have proposed only a simple algorithm. The points $\mathbf{x}_i$ follow their gradient $\tilde{N}\mathbf{f}(\mathbf{x}_i)$ to find the new positions. This method is similar to particle systems approaches, [15], but it has an opposite order of steps; the triangulation is created at first and then the points are projected onto the implicit surface.

The algorithm is similar to the straight root finding algorithm, see Algorithm 9, with difference that the surface normal vector is computed in each step.

Let the initial mesh be created. Then the next procedure continues as follows.

Algorithm 11. Projecting the points of an initial mesh onto the original implicit surface.

1. Set $\varepsilon = 0$.
2. For each point $\mathbf{x}_i$ compute its new normal vector $\mathbf{n}_i$.
3. Move the point to its new position $\mathbf{x}_i$' in the normal vector direction,

   $\mathbf{x}_i' = \mathbf{x}_i + \delta * \text{sign}(f_i)\, \mathbf{n}_i$,

   where δ is a step and $\text{sign}(f_i)$ is the signum function of the function value f_i in the point $\mathbf{x}_i$.
4. Determine function values f_i, f_i' at points $\mathbf{x}_i$, $\mathbf{x}_i'$.
5. Check next two cases.
   a) If these points lie on opposite sides of implicit surface, i.e. $(f_i * f_i') < 0$; compute the exact coordinates of the point $\mathbf{x}_i$ by binary subdivision between these points.
   b) If the points $\mathbf{x}_i$, $\mathbf{x}_i'$ lie on the same side of the surface then $\mathbf{x}_i = \mathbf{x}_i'$ and return to step 2.

3.4.1. Experimental results

In this section, several complex implicit objects with sharp features will be properly polygonized by the Adaptive Edge spinning approach using the technique proposed above. Triangulations of some examples are shown in Figure 57 and corresponding formulas can be found in Appendix C.

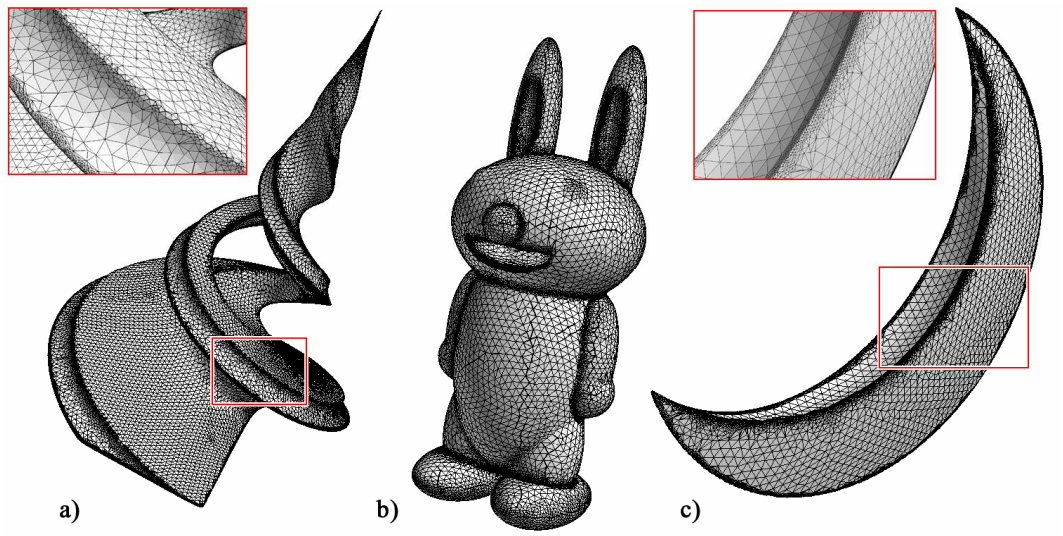


Figure 57. Objects generated by the Adaptive edge spinning algorithm, with usage of the introduced technique, a) the Yutaka object - taken from [27], b) the Rabbit modeled by the F-Rep [22] and c) the Eclipse model.

Figure 58 shows the implicit model of a tap with its normal vectors array. The Tap object consists of sharp edges as well.

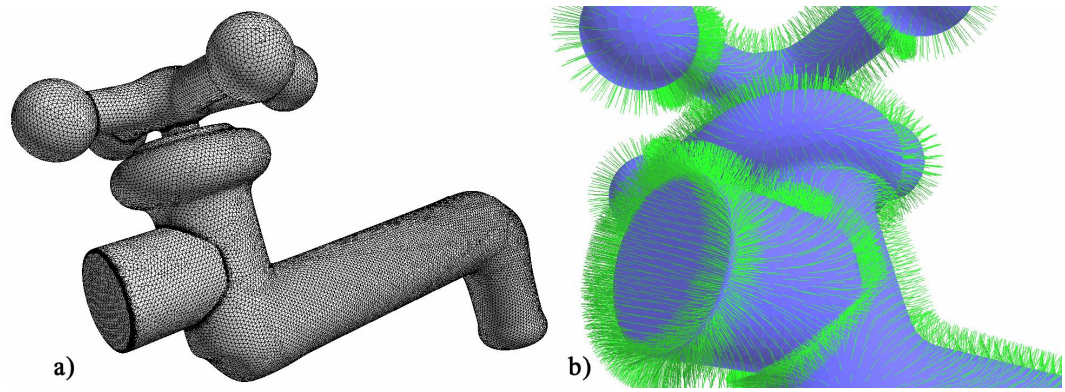


Figure 58. A tap generated by the Adaptive edge spinning algorithm; also with usage of the introduced trick, a) its triangulation, b) the array of normal vectors.

Note that the objects Tap and Rabbit have been modeled with use of the implicit modeling module, [41], which is a part of the MVE (Modular Visualization Environment) developed at University of West Bohemia, [25].

Table 7 contains values measured on complex implicit objects visualized in figures above. It is obvious that the Taubian distance is not good enough for such complex objects as the approximation of the real distance.

Edge spinning	yutaka	rabbit	eclipse	tap
e	0,1	0,1	6,0	0,1
LOD	0,32	0,32	0,32	0,32
Triangles	111 173	48 529	31 233	38 184
Vertices	55 648	24 268	15 627	19 094
Angle err	5,31E-02	4,55E-02	6,94E-02	4,40E-02
Centroid angle err	1,43E-02	1,06E-02	3,15E-02	1,00E-02
Alg dist avg	9,76E-02	9,79E-02	9,32E-01	9,37E-02
Euc dist avg	1,42E-03	2,06E-03	1,11E-03	3,03E-03
Taub dist avg	8,85E-02	1,77E-01	2,29E-02	6,10E-02
Angle criterion	0,649	0,662	0,618	0,684
Edge length criterion	0,772	0,780	0,748	0,796
Time [ms]	10 175	7 841	6 519	7 741
Time avg [ms]	91,52	161,57	208,72	202,72

Table 7. Values generated by the Edge spinning algorithm.

Note that the average time values for creating of one thousand triangles are higher in comparison with the simpler models because the one call of the function takes more time.

3.4.2. Conclusion

The Edge spinning algorithm can polygonize variety of implicit surfaces whose size and degree of continuity is not known ahead. It is possible with use of the introduced technique when the epsilon value instead of the zero value of an implicit function is used for computation. In that case, the implicit model has better differential properties.

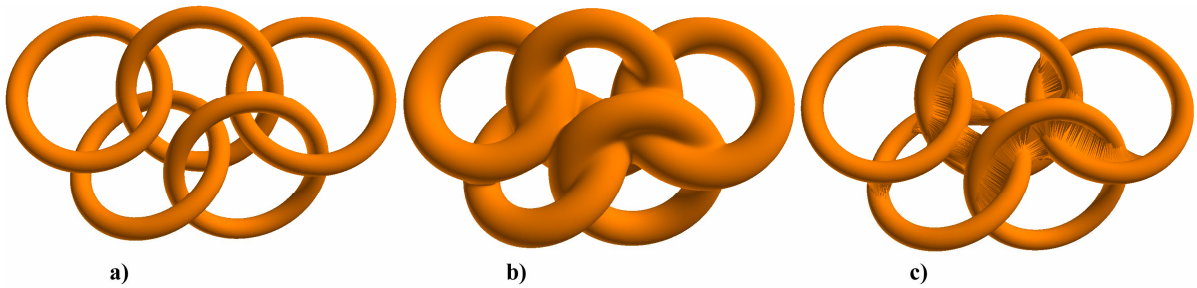


Figure 59. a) the original Olympic Rings object b) polygonized with ε = 10 and c) visualization after projection back to ε = 0.

Of course there must be a limitation. There is no exact way how to predict value of epsilon. It depends on a size of an object, sharpness of edges, etc. Moreover, for higher

epsilon, an implicit object could change its topology, see Figure 59. In such cases, the projection phase cannot work properly.

The presented approaches in this section have been sent for publication, see [xi] of the author publications.

3.5. Detection and polygonization of disjoint implicit surfaces in a given area

In this section, a new method how to detect, count, and polygonalize more disjoint implicit surfaces will be introduced. The algorithm uses the Edge spinning method for polygonization of each component, so there is necessary to detect a starting point for each of them.

Because of an implicit function can be an arbitrary unknown algebraic function, there is no other way how to detect more disjoint surfaces in a defined area than use of exhaustive search approach (described for the Marching cubes and tetrahedra methods in section 2.3).

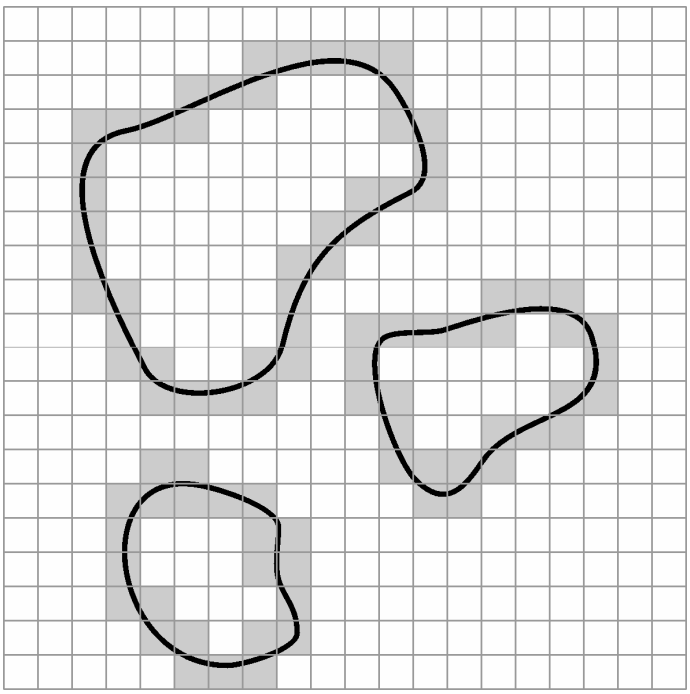


Figure 60. The polygonization area divided by the regular grid that contains three implicit objects. Grid cells intersected by the implicit function are highlighted.

Our algorithm divides the given polygonization area by the regular grid. An important note is that a size of grid cells need not to be proportional to extracted object detail but it should only be proportional to the size of the smallest object that is wanted to be

visualized. The size of grid cells is only needed for detection of implicit components in a scene, it has no relation to object detail and this is the main difference from the Marching cubes approaches. Therefore, a number of grid divisions is much lesser for our algorithm than for Marching cubes method as well as a computational time.

Let the polygonization area be defined in space as [-x, +x ; -y, +y ; -z, +z] and a number of division in each axis be ***M***. Then the algorithm works as follows.

Algorithm 12. Polygonization of more disjoint surfaces in defined area.

1. Use the Edge spinning algorithm to polygonalize the first object in the area.
2. Create a function grid – find and mark all grid cells intersected by the implicit function.
3. Create a triangulation grid – find and mark all grid cells intersected by the triangular mesh.
4. Check if there is a marked function grid cell that has an unmarked equivalent and even unmarked neighborhood in the triangulation grid.
   a) If YES – there is a new implicit component and continue as follows.
      - Find a new starting point in the given cell.
      - Use the ES method for triangulation of the component.
      - Return to step 3.
   b) If NO – there is no other component – end of polygonization.

Notes:

- step 2 – a grid cell is marked if at least two of their corners have opposite signs of the function; important note is that this step is performed just once and in case of complex functions it saves computational time
- step 3 – if the size of grid cells is greater than or equal to the longest edge of a triangle then it is enough to marked a grid cell when a point of a triangle is located in there
- step 4 – the neighborhood in E^3 means 26 adjacent grid cells to the given one
- step 4a – a new starting point in the given cell is sought by the binary subdivision between two cell's corners with opposite signs.

3.5.1. Experimental results

Experimental results in this section are aimed at polygonization of unknown implicit scenes consisting of more disjoint surface components. The Adaptive Edge spinning algorithm (AES) will be compared with the Marching cubes method – exhaustive search (MCE).

The first scene is illustrated in Figure 61 and contains two entwined spirals.

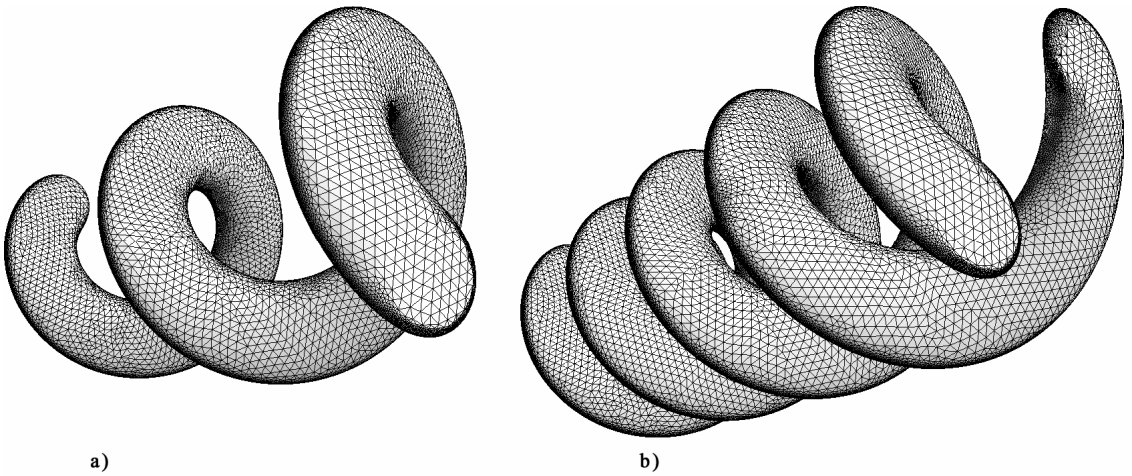


Figure 61. The Spirals model polygonized by the Adaptive Edge spinning algorithm; a) the first spiral and b) both spirals polygonized.

Regarding to recent Olympic Games, the second implicit scene represents Olympic rings. The model consists of five disjoint components and has been modeled as union of five tori, see Figure 62.

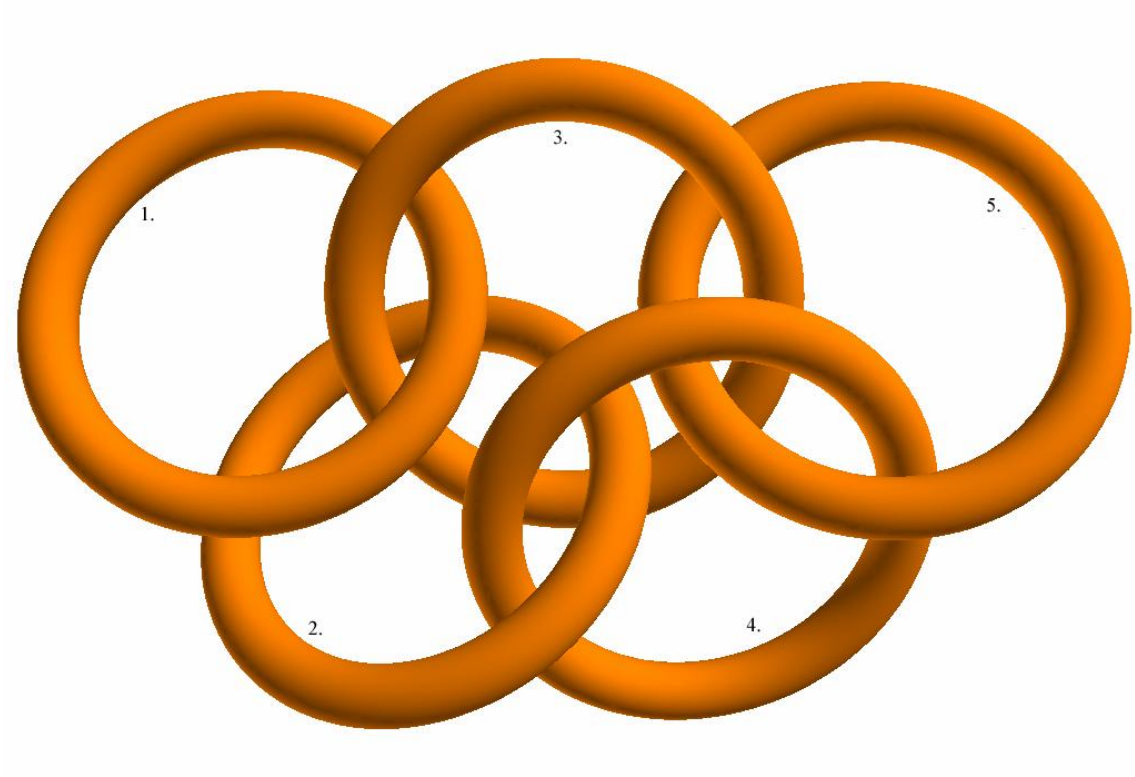


Figure 62. The Olympic rings model polygonized by the Adaptive Edge spinning algorithm. The numbers means the order of polygonization.

	Adaptive Edge spinning		Marching cubes	
	Spiral	Olympic rings	Spiral	Olympic rings
Subdivisions	50	50	400	300
LOD	0,16	0,16	0,08	0,11
Angle error set	5,00E-02	5,00E-02	---	---
Triangles	134 316	147 346	131 776	230 572
Verices	67 163	73 673	65 892	115 286
Angle error	3,11E-02	2,97E-02	2,45E-02	2,51E-02
Centroid angle error	7,01E-03	6,96E-03	8,10E-03	7,48E-03
Alg dist avg	1,79E-03	1,88E-02	2,65E-03	1,51E-02
Euc dist avg	5,91E-04	1,47E-03	8,70E-04	1,18E-03
Taub dist avg	5,93E-04	1,47E-03	8,76E-04	1,18E-03
Angle criterion	0,70	0,69	0,36	0,37
Edge length criterion	0,81	0,80	0,52	0,53
Time [ms]	6 453	7 375	42 844	27 422
Avg time [ms]	48,04	50,05	325,13	118,93

Table 8. Values generated by the Edge spinning and the Marching cubes algorithms.

The measured values from the experiment are contained in Table 8. The level of detail for both algorithms has been set so that a number of triangles as well as the approximation quality to be similar. In such case, the Marching cubes method is much slower then the Edge spinning algorithm and moreover, it generates poor triangular mesh, see histogram of angle distributions in Figure 63.

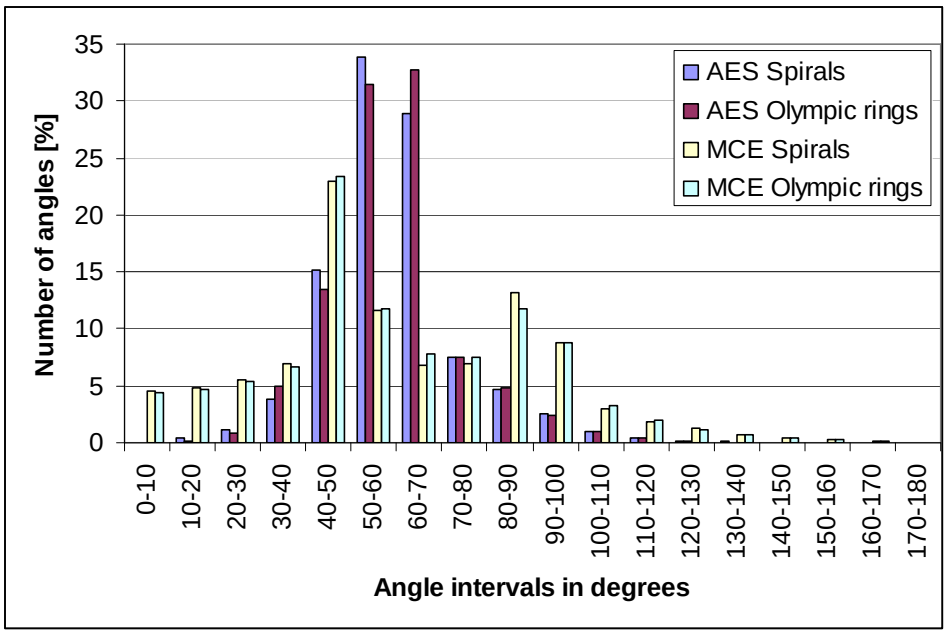


Figure 63. Histogram of triangles shape quality.

3.5.2. Conclusion

The presented approach in this section makes possible polygonization of more disjoint implicit surfaces in a defined area. The algorithm is not limited to a given polygonization method but then, its use is possible for all other surface approaches that need a starting point at the beginning. The algorithm works well, it is able to found and polygonize all implicit components in a given region and the computational time is much less than in case of Marching cubes method – exhaustive search. All advantages of surface approaches are preserved.

4. Conclusion and future work

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 and are not able to polygonize implicit objects consisting of more disjoint surfaces.

In the presented dissertation thesis, there are several new approaches for polygonization of implicit surfaces introduced, based on the Edge spinning principle. The first of them is the non-adaptive algorithm that generates triangular meshes of high quality and its polygonization speed is comparable with the well-known Marching cubes algorithm.

The adaptive modification of the given approach has been developed in the next research. The algorithm generates triangles of size according to the local surface curvature estimation. The curvature is estimated by deviation of surface normal vectors. This technique is simple and fast for computation as well as effective in results. A size of triangles varies to preserve the approximation error given at the beginning of polygonization and this is the main advantage in comparison to other methods. The whole process is directed to achieve the given accuracy and the algorithm maintains this requirement in all places of an implicit object (high/low curvature). It means that the resulting polygonal mesh does not consists only of well-shaped triangles, but moreover, the mesh satisfies predefined requirements of accuracy as well.

The Edge spinning algorithm has a common deficiency of surface approaches, i.e. it is sensitive to implicit functions of only C^0 continuity. A computation of a gradient vector (normal vectors) in areas of sharp features is influenced by a major error and surface approaches become unstable in such regions. Therefore, we have proposed a simple technique how to bypass this problem. The method depends on modeling technique F-Rep that gives to resulting objects good differential properties.

The last part of the thesis is aimed at implicit scenes consisting of more disjoint surfaces. In case of unknown implicit functions, there is only the volume based Marching cubes algorithm – exhaustive search able to polygonize all surfaces in the defined area. Surface approaches need a starting point for computation and such point

has to be found for each disjoint component. We have designed such algorithm and unknown complex scenes can be triangulated by surface approaches of higher quality of details and much faster then in case of use of exhaustive search now.

In our future research, we want to improve the ability of the Edge spinning algorithm to polygonize implicit objects of C^0 continuity with proper edge detection and edge extraction in the resulting polygonal mesh.

References

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

[2] Allgower, E.L., Gnutzmann, S.: An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces. SIAM Journal of Numerical Analysis, 24, 452-469, April 1987.

[3] Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., Desbrun, M.: Anisotropic Polygonal Remeshing, Siggraph 2003, ACM TOG, Volume 22 , Issue 3, 2003.

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

[5] Blinn, J.: A Generalization of Algebraic Surface Drawing. ACM Transactions on Graphics, 1(3):235–256, 1982.

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

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

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

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

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

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

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

[13] 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.

[14] Figueiredo, L.H.: Computational Morphology of Implicit Curves, doctoral thesis, IMPA, 1992.

[15] Figueiredo L.H., Gomes J.M., Terzopoulos D., Velho L.: Physically-based methods for polygonization of implicit surfaces, In Proceedings of Graphics Interface 92, 1992.

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

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

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

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

[20] 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.

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

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

[23] Karkanis, T., Stewart, A.J.: Curvature-Dependent Triangulation of Implicit Surfaces, IEEE Computer Graphics and Applications, Volume 21, Issue 2, March 2001

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

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

[26] 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.

[27] 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.

[28] 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.

[29] Pasko, A., Adzhiev, V., Karakov, M., Savchenko, V.: Hybrid system architecture for volume modeling, Computer & Graphics 24 (67-68), 2000.

[30] 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.

[31] Pasko, A., Sourin, A., Savchenko, V.: Using Real Functions with Application to Hair Modeling, C&G(20), 1996, pp. 11-19.

[32] Rektorys, K.: Přehled užité matematiky, the textbook of mathematics (in Czech), SNTL 1981.

[33] Rousal, M., Skala, V.: Modular Visualization Environment - MVE, Int. Conf. ECI 2000, Herlany, Slovakia, pp.245-250, ISBN 80-88922-25-9.

[34] Rvachev, A.M.: Definition of R-functions, http://www.mit.edu/~maratr/rvachev/p1.htm

[35] Shapiro, V., Tsukanov, I.: Implicit Functions with Guaranteed Differential Properties, Solid Modeling, Ann Arbor, Michigan, 1999.

[36] Shu R., Zhou, Ch., Kankanhalli, M.S.: Adaptive Marching Cubes, The Visual Computer, 11: 202-217, 1995.

[37] Stoker, J.J.: Differential geometry, New York: Willey-Interscience, ISBN 0-471-50403-3, 1989.

[38] 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.

[39] Taubin, G.: Distance Approximations for Rasterizing Implicit Curves, ACM Transactions on Graphics, January 1994.

[40] Triquet, F., Meseure, F., Chaillou, Ch.: Fast Polygonization of Implicit Surfaces, WSCG 2001 Int.Conf., pp. 162, University of West Bohemia in Pilsen, 2001.

[41] Uhlíř, K., Skala, V.: Kompilovaný HyperFun, Research report (in Czech) No. DCSE/TR-2002-07, University of West Bohemia, 2002.

[42] Velho, L.: Simple and Efficient Polygonization of Implicit Surfaces, Journal of Graphics Tools, Volume 1 Number 1, 1996, pp. 5-24.

[43] Velho, L., Gomes, J., Figueiredo, L.H.: Implicit Objects in Computer Graphics, Springer, ISBN 0-387-98424-0, 2002.

[44] Wyvill, B., Liang, X.: Hierarchical Implicit Surface Refinement, International Conference CGI, 2001.

[45] 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.

[46] 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.

[47] Wyvill, B., Galbraith, C., MacMurchy, P.: BlobTree Trees, International Conference CGI, 2004.

[48] 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.

[49] 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

List of publications

[i] Čermák,M., Skala,V. Adaptive Edge Spinning Algorithm for Polygonization of Implicit Surfaces. Computer Graphics International, CGI 2004, IEEE, Crete, Greece, 2004.

[ii] Čermák,M., Skala,V. Surface Curvature Estimation for Edge Spinning Algorithm. International Conference on Computational Science, ICCS 2004, Springer, Krakow, Poland, 2004.

[iii] Čermák,M., Skala,V. Adaptive Edge Spinning Algorithm for Implicit Surfaces. International Conference on Computational Science and its Applications, ICCSA 2004, Springer, Assisi, Italy, 2004.

[iv] Čermák,M., Skala,V. Edge spinning algorithm for implicit surfaces, Journal of Applied Numerical Mathematics, Volume 49, Issues 3-4, Pages 331-342, Elsevier, June 2004.

[v] Čermák,M., Skala,V. Detection of Sharp Edges during Polygonization of Implicit Surfaces by the Edge Spinning. Geometry and graphics in teaching contemporary engineer, Szczyrk 2003, Poland, June 12-14, 2003.

[vi] Čermák,M., Skala,V. (supervisor). Methods for Implicit Surfaces Polygonization. State of the Art and Concept of Doctoral Thesis, Technical report No. DCSE/TR-2003-01, University of West Bohemia, Plzen, Czech Republic, 2003.

[vii] Čermák, M., Skala, V. Space Subdivision for Fast Polygonization of Implicit Surfaces. The Fifth International Scientific Conference, ECI 2002, Slovakia, pp. 302-307, October 10-11, 2002.

[viii] Čermák, M., Skala, V. Polygonization by the Edge Spinning. 16th Conference on Scientific Computing, Algoritmy 2002, Slovakia, pp. 245-252, September 8-13, 2002.

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

[x] 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.

Submitted for publication:

[xi] Čermák, M., Skala, V. Polygonization of Implicit Surfaces with Sharp Features by the Edge Spinning, The Visual Computer journal, May 2004.

[xii] Čermák, M., Skala, V. Polygonization of Disjoint Implicit Surfaces by the Adaptive Edge Spinning Algorithm, International Journal of Computational Science and Engineering, Inderscience Publishers, November 2004.

Appendix B

Stays and Lectures Abroad

Stays

15.2.2001	-	15.6.2001	University of Lisboa, Portugal, Erasmus/Socrates project

Conferences

29.8.2004	-	3.9.2004	Eurographics 2004, Grenoble, France	
16.6.2004	-	19.6.2004	CGI 2004, Crete, Greece	[i]
6.6.2004	-	9.6.2004	ICCS 2004, Krakow, Poland	[ii]
14.5.2004	-	17.5.2004	ICCSA 2004, Assisi, Italy	[iii]
12.6.2003	-	14.6.2003	Szczyrk 2003, Poland	[v]
10.10.2002	-	11.10.2002	ECI 2002, Herlany, Slovakia	[vii]
25.9.2002	-	29.9.2002	ICCVG 2002, Zakopane, Poland	[ix]
8.9.2002	-	13.9.2002	Algoritmy 2002, Podbanske, Slovakia	[viii]
1.9.2002	-	6.9.2002	Eurographics 2002, Saarbrücken, Germany	

Publications

See Appendix A and E for details.

Funding

The work was supported by the Ministry of Education of the Czech Republic - project MSM 235200005.

Appendix C

Implicit functions codes

Boolean operations – F-Rep

```
double unionAB(double A, double B)
{
      return (A + B + sqrt( A*A + B*B );
}
double intersectionAB(double A, double B)
{
      return (A + B - sqrt( A*A + B*B );
}
double differencesAB(double A, double B)
{
      return intersectionAB(A, -B);
}
```

Sphere

```
double common_sphere(double c1, double c2, double c3, double rad,
       double x, double y, double z) {
  double s, a1,a2,a3;
  a1 = (c1-x)*(c1-x);
  a2 = (c2-y)*(c2-y);
  a3 = (c3-z)*(c3-z);
  s = rad*rad - a1 - a2 - a3;
  return s;
}
```

Eclipse

```
double eclipse(double x, double y, double z)
{
  double s1a = common_sphere(-30.0,8.0,0.0, 31.0, x, y, z);
  double s1b = common_sphere( 30.0,8.0,0.0, 31.0, x, y, z);
  double s1c = common_sphere( 0.0,10.0,-3.0, 7.0, 2.0*x, y, z);
  double s1d = intersectionAB(s1a,s1b);
  double s1  = differencesAB(s1d,s1c);
  return (s1);
}
```

Genus

```
double genus(double x, double y, double z)
{
```

```
  double rx = 6, ry = 3.5, rz = 4;
  double r1 = 1.2, x1 = 3.9;
  double y2 = SQR(y);
  double r12 = SQR(r1);
  double g1 = SQR(rz)*SQR(rz)*SQR(z);
  double g2 = ( 1 - SQR(x/rx) - SQR(y/ry) );
  double g3 = ( SQR(x-x1) + y2 - r12 ) * ( x*x + y2 - r12 );
  double g4 = ( SQR(x+x1) + y2 - r12 );
  return (-g1 + g2*g3*g4);
}
```

Yutaka

```
double yutaka(double x, double y, double z) {
  float x1 = (float)x;
  float y1 = (float)y;
  float z1 = (float)z;
  float tar = (y1+15.0f)/15.0f;
  x1 /= tar*tar;
  z1 /= tar*tar;
  double angle = 4*PI*y1/30.0f;
  float xt = x1*cos(angle) + z1*sin(angle);
  float yt = y1;
  float zt = -x1*sin(angle) + z1*cos(angle);
  float rect = intersectionAB(intersectionAB(3 - fabs(xt), 15 -
       fabs(yt)), 1 - fabs(zt));
  float rect3 = 4 - xt*xt - (yt-5)*(yt-5)/4;
  x1 = x; y1 = y; z1 = z;
  tar = (y1+14)/10;
  x1 /= tar*tar;
  z1 /= tar*tar;
  float sphere = 5 - x1*x1 - (y1+10)*(y1+10) - z1*z1;
  float f1 = intersectionAB(rect, -rect3);
  float f2 = sphere;
  float fff = unionAB(f1, f2) + 100/(1+(f1/2)*(f1/2)+(f2/2)*(f2/2));
  return (fff);
}
```

Torus

```
double torus (double x, double y, double z, double R, double r)
{
  double x2 = x*x, y2 = y*y, z2 = z*z;
  double a = x2+y2+z2+(R*R)-(r*r);
  return -a*a+4.0*(R*R)*(x2+y2);
}
```

Olympic rings

```
double olympic_rings(double x, double y, double z)
{
  double s = 0.5; // scale
  double s1 = 1.0 / s;
  x =  s * x; y =  s * y; z =  s * z;
  double R=1.4*s1;
  double r=0.2*s1;
  double a, b, c, d, e, f;
  double y2,z2;
```

```
  POINT1 p1, p2, p3, p4; // struct with three double x, y, z
  p1 = rotation_x(-15.0, x, y, z); // angle in degrees
  p2 = rotation_x( 20.0, x, y, z);
  p3 = rotation_x( 30.0, x, y, z);
  p3 = rotation_y(-10.0, p3);
  p4 = rotation_x(-30.0, x, y, z);
  p4 = rotation_y( 15.0, p4);
  a = torus(p1.x-2.1*s1, p1.y, p1.z, R, r);
  b = torus(p2.x, p2.y, p2.z, R, r);
  c = torus(p1.x+2.1*s1, p1.y, p1.z, R, r);
  d = torus(p3.x-0.8*s1, p3.y+1.3*s1, p3.z-0.8*s1, R, r);
  e = torus(p4.x+1.1*s1, p4.y+1.1*s1, p4.z+0.4*s1, R, r);
  f = unionAB( unionAB(a,b), unionAB(c,d) );
  return ( unionAB(e,f) );
}
```

Cube

```
double cube(double x, double y, double z)
{
  double r = 0.5;
  double a, b, c, d;
  a = -fabs(x) + r;
  b = -fabs(y) + r;
  c = -fabs(z) + r;
  d = intersectionAB(a,b);
  return intersectionAB(c, d);
}
```

Jack

```
double jack(double x, double y, double z)
{
  double x2 = x*x, y2 = y*y, z2 = z*z;
  double a1 = x2/9 + 4*y2 + 4*z2;
  double a2 = y2/9 + 4*x2 + 4*z2;
  double a3 = z2/9 + 4*y2 + 4*x2;
  double b11 = 4*x/3-4;
  double b12 = 4*x/3+4;
  double b21 = 4*y/3-4;
  double b22 = 4*y/3+4;
  double c1 = b11*b11 +16*y2/9 + 16*z2/9;
  double c2 = b12*b12 +16*y2/9 + 16*z2/9;
  double c3 = b21*b21 +16*x2/9 + 16*z2/9;
  double c4 = b22*b22 +16*x2/9 + 16*z2/9;
  double m1 = 1/(a1*a1*a1*a1) + 1/(a2*a2*a2*a2) + 1/(a3*a3*a3*a3);
  double m2 = 1/(c1*c1*c1*c1) + 1/(c2*c2*c2*c2) + 1/(c3*c3*c3*c3) +
       1/(c4*c4*c4*c4);
  double m3 = 1/sqrt(sqrt(m1+m2));
  return(-m3+1);
}
```

Spiral

```
double spiral(double xx, double yy, double zz)
{
  double x, y, z;
```

```
  x = xx; y = yy; z = zz;
  double R = 2.0;
  double r = 0.8;
  double cx = R * cos(z);
  double cy = R * sin(z);
  double sp = r*r - (x+cx)*(x+cx) - (y+cy)*(y+cy);
  double cube = uni_cube(x, y, z, 7.0); // clipping by a cube
  double spd = intersectionAB(sp,cube);
  return (spd + 0.2);
}
```

Spirals

```
double spirals(double x, double y, double z)
{
  double s1 = spiral (x, y, z);
  POINT1 p;
  p.x = x; p.y = y; p.z = z;
  p = rotation_z(180.0, p);
  double s2 = spiral(p.x, p.y, p.z);
  double ss = unionAB(s1, s2);
  return ss;
}
```

Morph

```
double morph (double x, double y, double z)
{
  double r = 2.2; // morphing ratio
  double o1 = common_sphere(-0.2, 0, 0, 1, x, y, z);
  double o2 = jack(x, y, z);
  double m = r * o1 + (1.0 - r) * o2;
  return (m);
}
```

Note that following two functions are modeled by the Hyperfun modeling module [41], so they have different syntax.

Rabbit

```
FmodelDouble FmodelDouble::rabbit(double x[])
{
double center[3];
double xt[3];
FmodelDouble kao,head,body;
FmodelDouble kao1,sp1,rmimi,r2mimi,lmimi,l2mimi,mimi;
FmodelDouble hana,reye,leye,mouth1,face1,mouth;
FmodelDouble lleg,rleg,lfoot,rfoot,ashi,tail,lude,
       rude,lhand,rhand,ude;
xt[1] = x[0];
xt[2] = x[1];
xt[3] = x[2];
center[0] = 0.0;center[1] = 0.0;center[2] = 1.5;
kao1 = fEllipsoid(x,center,4,4,3);
center[0] = 0.0;center[1] = 1.4;center[2] = 1.5;
```

```
sp1 = fSphere(x,center,5);
kao = kao1 & sp1;
center[0] = -2.0;center[1] = -1.0;center[2] = 5.5;
rmimi = fEllipsoid(x,center,1,1,4);
center[0] = -2.0;center[1] = 0.0;center[2] = 6.0;
r2mimi = fEllipsoid(x,center,0.5,0.5,2);
center[0] = 2.0;center[1] = -1.0;center[2] = 5.5;
lmimi = fEllipsoid(x,center,1,1,4);
center[0] = 2.0;center[1] = 0.0;center[2] = 6.0;
l2mimi = fEllipsoid(x,center,0.5,0.5,2);
mimi = (rmimi % r2mimi) | (lmimi % l2mimi);
center[0] = 0.0;center[1] = 4.2;center[2] = 1.5;
hana = fSphere(x,center,0.7);
center[0] = -2.0;center[1] = 2.5;center[2] = 3.0;
reye.m_dRes = fSphere(x,center,0.5);
center[0] = 2.0;center[1] = 2.5;center[2] = 3.0;
leye = fSphere(x,center,0.5);
center[0] = 0.0;center[1] = 3.0;center[2] = 0.8;
mouth1 = fEllipsoid(x,center,2,2,1);
face1 = 0.8-x[2];
mouth = mouth1 & face1;
head = (kao | mimi | hana | reye | leye) % mouth;
center[0] = 0.0;center[1] = 0.0;center[2] = -4.0;
body = fEllipsoid(x,center,3,3,5);
center[0] = -1.7;center[1] = 0.0;center[2] = -8.0;
lleg = fEllipsoid(x,center,1.5,1.5,2.7);
center[0] = 1.7;center[1] = 0.0;center[2] = -8.0;
rleg = fEllipsoid(x,center,1.5,1.5,2.7);
center[0] = 2.0;center[1] = 0.2;center[2] = -10.0;
lfoot = fEllipsoid(x,center,1.8,3,1);
center[0] = -2.0;center[1] = 0.2;center[2] = -10.0;
rfoot = fEllipsoid(x,center,1.8,3,1);
ashi = lleg | rleg | lfoot | rfoot;
center[0] = 0.0;center[1] = -3.0;center[2] = -7.0;
tail = fSphere(x,center,0.8);
center[0] = -3.0;center[1] = 0.0;center[2] = -3.5;
lude = fEllipsoid(x,center,1,1,2.5);
center[0] = 3.0;center[1] = 0.0;center[2] = -3.5;
rude = fEllipsoid(x,center,1,1,2.5);
center[0] = -3.2;center[1] = 0.0;center[2] = -5.0;
lhand = fEllipsoid(x,center,1,1,1);
center[0] = 3.2;center[1] = 0.0;center[2] = -5.0;
rhand = fEllipsoid(x,center,1,1,1);
ude = lude | rude | lhand | rhand;
return(head | body | ashi | tail | ude);
}
```

Tap

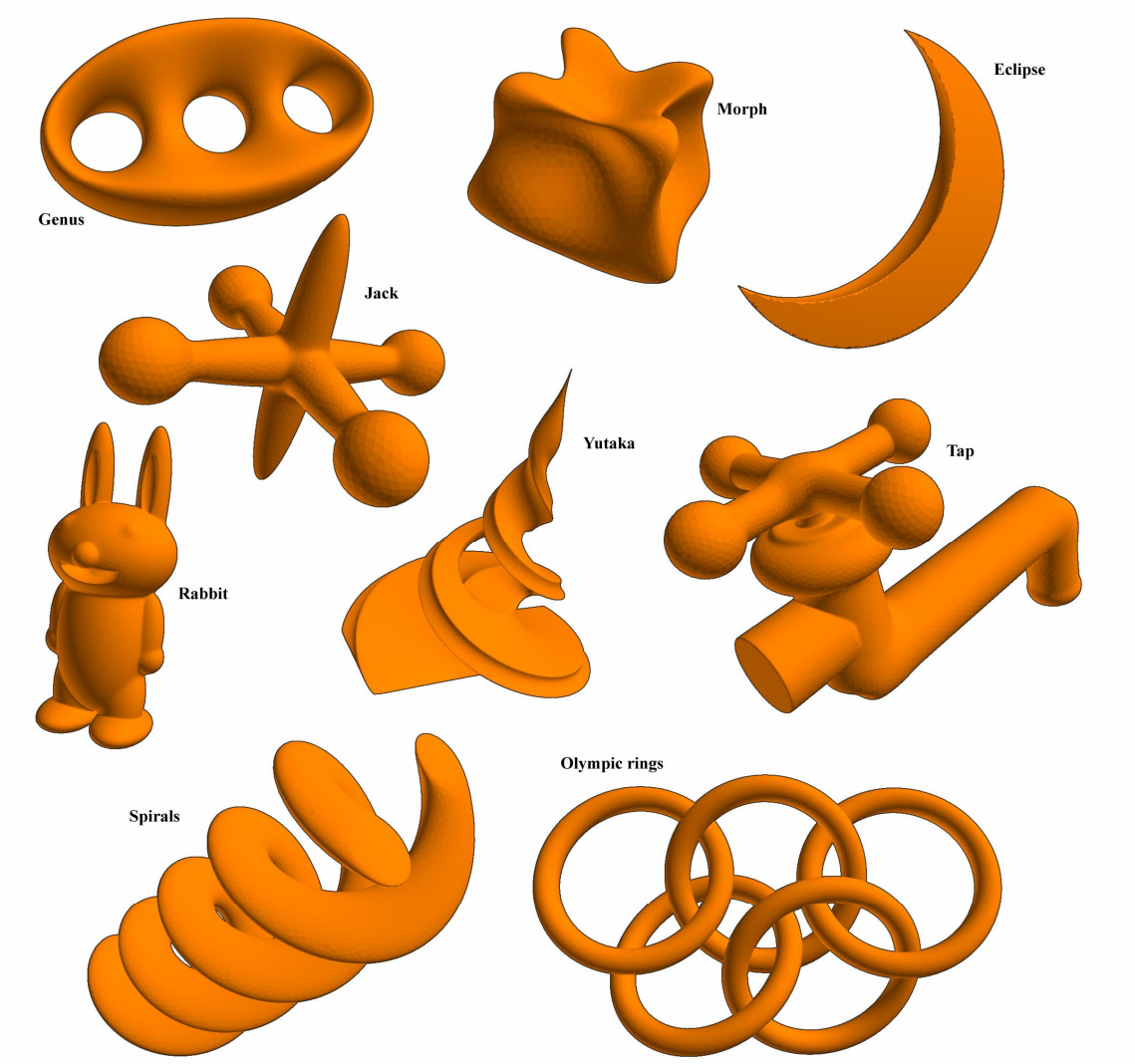
```
FmodelDouble FmodelDouble::tap(double x[])
{
 double center1[3],center2[3],center3[3],center4[3],
       center5[3],center6[3];
 double center7[3],center8[3],center9[3],center10[3],center11[3];
 double xt,yt,zt;
 FmodelDouble theta,cyl1,cyl2,cyl3,cyl4,bluni1,bluni2;
```

```
FmodelDouble cyl5,cyl6,torus1,torus2,pipe1,pipe,cyl7,cyl8,
      grip1,sp1,sp2,sp3;
FmodelDouble cyl1a,cyl1b,cyl1c,cyl2a,cyl2b,cyl2c,cyl3a,
      cyl3b,cyl3c,cyl4a,cyl4b,cyl4c;
FmodelDouble cyl5a,cyl5b,cyl5c,cyl6a,cyl6b,cyl6c,cyl7a,
      cyl7b,cyl7c,cyl8a,cyl8b,cyl8c;
FmodelDouble sp4,grip,rotatey1;
xt = x[0]; yt = x[1]; zt = x[2];
theta = 0.25*x[3] +3.5;
center1[0] = 1; center1[1] = 0; center1[2] = 0;
center2[0] = -7; center2[1] = 0; center2[2] = 0;
center3[0] = 10; center3[1] = 0; center3[2] = 0;
center4[0] = 0; center4[1] = 3; center4[2] = 0;
center5[0] = -7; center5[1] = 7; center5[2] = 0;
center6[0] = 10; center6[1] = -6; center6[2] = 0;
center7[0] = -7; center7[1] = 10; center7[2] = 0;
center8[0] = -1; center8[1] = 10; center8[2] = 0;
center9[0] = -12; center9[1] = 10; center9[2] = 0;
center10[0] = -7; center10[1] = 10; center10[2] = 6;
center11[0] = -7; center11[1] = 10; center11[2] = -6;
// --- pipe ---
cyl1a = yt;
cyl1b = -yt+8;
cyl1c = fCylinderY(x,center2,2);
cyl1 = cyl1a & cyl1b & cyl1c;
cyl2a = (xt+7);
cyl2b = (-xt+9);
cyl2c = fCylinderX(x,center1,2);
cyl2 = cyl2a & cyl2b & cyl2c;
cyl3a = (yt+6);
cyl3b = (-yt-1);
cyl3c = fCylinderY(x,center3,1.7);
cyl3 = cyl3a & cyl3b & cyl3c;
cyl4a = (xt+12);
cyl4b = (-xt-8);
cyl4c = fCylinderX(x,center4,2);
cyl4 = cyl4a & cyl4b & cyl4c;
bluni1 = fBlendUni(cyl1,cyl2,2,2,1);
bluni2 = fBlendUni(bluni1,cyl3,3,5,3);
cyl5a = yt;
cyl5b = (-yt+10);
cyl5c = fCylinderY(x,center2,0.7);
cyl5 = cyl5a & cyl5b & cyl5c;
cyl6a = (xt+9.5);
cyl6b = (-xt-10);
cyl6c = fCylinderX(x,center4,5);
cyl6 = cyl6a & cyl6b & cyl6c;
torus1 = fTorusY(x,center5,2.3,1);
torus2 = fTorusY(x,center6,1,0.8);
pipe1 = fBlendUni(bluni2,torus1,1,1,1);
pipe = pipe1|torus2;
// --- grip ---
cyl7a = (xt+12);
cyl7b = (-xt-1);
cyl7c = fCylinderX(x,center7,0.8);
cyl7 = cyl7a & cyl7b & cyl7c;
cyl8a = (zt+6);
```

```
 cyl8b = (-zt+6);
 cyl8c = fCylinderZ(x,center7,0.8);
 cyl8 = cyl8a & cyl8b & cyl8c;
 grip1 = fBlendUni(cyl7,cyl8,2,1,1);
 sp1 = fSphere(x,center8,1.5);
 sp2 = fSphere(x,center9,1.5);
 sp3 = fSphere(x,center10,1.5);
 sp4 = fSphere(x,center11,1.5);
 rotatey1 = fRotate3DY(center7,45);
 grip = grip1|sp1|sp2|sp3|sp4;
 return(bluni1|bluni2|cyl4|cyl5|cyl6|pipe|grip);
}
```

Appendix D

Implicit functions pictures



Appendix E

Publications