Visual Comput (2005) 00: 1–13
DOI 10.1007/s00371-005-0286-2

ORIGINAL ARTICLE

Martin Čermák[1] CE[a]
Václav Skala

Polygonization of implicit surfaces with sharp features by edge-spinning

Received: ?? ? ????
Accepted: ?? ? ????
Published online: ?? ? ????

Martin Čermák · Václav Skala
University of West Bohemia in Pilsen,
Department of Computer Science and
Engineering, Czech Republic
{cermakm|skala}@kiv.zcu.cz

Abstract This paper presents an adaptive approach for polygonization of implicit surfaces. The algorithm generates a well-shaped triangular mesh with respect to a given approximation error. The error is proportional to a local surface curvature estimation. Polygonization of surfaces of high curvature, as well as surfaces with sharp features, is possible using a simple technique combined with a particle system approach. The algorithm is based on a surface tracking scheme, and it is compared with other algorithms based on a similar principle, such as the marching cube and the marching triangle algorithms.

Keywords implicit function · polygonization · curvature · sharp features

1 Introduction

Implicit surfaces seem to be one of the most appealing concepts used in building complex shapes and surfaces. They have become widely used in several applications in computer graphics and visualization.

An implicit surface is mathematically defined as a set of points $\boldsymbol{x}$ in space that satisfy the equation $f(\boldsymbol{x}) = 0$. Thus, implicit surface visualization typically consists of finding the zero set of f, which may be performed either by polygonizing the surface or by direct ray-tracing.

There are two different definitions of implicit surfaces. The first one [5, 6] defines an implicit object as $f(\boldsymbol{x}) < 0$ and the second one, F-rep [17, 22, 24], defines it as $f(x) \geq 0$. These inequalities describe a half-space in E^3, and an object defined by these inequalities is usually called a solid (or volume).

If f is an arbitrary procedural method (i.e., a "black box" function that evaluates $\boldsymbol{x}$), then the geometric properties of the surface can be deduced only through numerical evaluation of the function. The value of f is often a measure of distance between $\boldsymbol{x}$ and the surface. The measure is Euclidean if it is an ordinary (physical) distance. For an algebraic surface, f measures the algebraic distance.

Existing polygonization techniques may be classified into several categories. Spatial sampling techniques (exhaustive enumeration of a given region) regularly or adaptively sample the space to find the cells that straddle the implicit surface [2, 5, 7, 8, 18, 26] CE[b].

Surface tracking approaches (also known as continuation methods) iteratively create a triangulation from a seed element by marching along the surface [1, 5, 9, 10, 15, 16, 18, 29].

Surface-fitting techniques progressively adapt and deform an initial mesh to converge to the implicit surface [22, 32].

Particle systems (physically based techniques) start from initial positions in space and seek their equilib-

[1] Supported by the Ministry of Education of the Czech Republic – project MSM 235200005

CE[a] Consider moving the information in your footnote to the acknowledgement section

CE[b] Please check the changes to this sentence

rium positions – i.e., positions where a potential function $|f|$ is minimal – on an implicit surface [13, 14]. The desired polygonal approximation is then obtained by computing the Delaunay triangulation associated with the points.

2 Motivation

Many polygonization algorithms adaptively or non-adaptively create polygonal meshes without a proper definition of an approximation error that is required in the result; for example, in Akkouche and Galin, Čermák and Skala, and Hartmann [1, 11, 15]CE[b]. Usually, the algorithms allow the user to set a level of detail (minimum/maximum size of triangles, number of divisions in the axes, etc.), which is only somewhat related to the resulting approximation quality. The quality strongly depends on the ratio between the size of the implicit objects and the size of triangles, the size of computational area, etc.

Our algorithm defines the approximation error that is proportional to the surface curvature estimation (see variable α_{err} in Sect. 4). 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 preserves the given approximation quality as well.

The presented algorithm is based on a surface tracking scheme, and it is able to polygonize implicit objects with sharp features as well. The paper introduces a simple technique on how to solve the problem with sharp edges. The technique uses an implicit function property that depends on F-Rep modeling [17], i.e., its principle is independent of the given polygonization method. It can be used under the same circumstances as all of the others surface tracking approaches, which is a significant advantage of our solutionCE[b].

3 Data structures

A triangular mesh generated by an edge-spinning algorithm is kept in a winged-edge data structure [4], and therefore, the resulting mesh is complete with neighborhood among triangles. During polygonization, edges lying on a triangulation border are contained in the active edge list (AEL) and they are called "active edges". Each point of an active edge has two pointers to its left and right active edge (active edges are oriented, and so left and right directions define the active edges' orientationsCE[b]).

For faster evaluation of the detection of global overlap, the space subdivision acceleration technique, introduced in Čermák and Skala [11], is used. Therefore, each point in AEL also contains an index of a subarea, and each subarea has its own dynamically allocated list of points located inside. The subareas are implemented as an array, and each of them has a unique index.

4 Polygonization process

The algorithm is based on a 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 a point. Several disjoint surfaces can be polygonized from a starting point for each surface. The whole algorithm consists of the following steps:

Algorithm 1: The main polygonization loop.

1. Initialization of polygonization:
   a) Locate a starting point $\boldsymbol{p}_0$ on a surface, and create the first triangle T_0 (see Algorithm 2)CE[c].
   b) Insert edges (e_0, e_1, e_2) of the first triangle T_0 into the active edge list, see Fig. 1.
2. Use the first active edge e from the active edge list for next polygonization.
3. Update the AEL; remove the currently polygonized active edge(s) and insert the new generated active edge(s) at the end of the list.
4. If the AEL is not empty, return to step 2.

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

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

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

All steps of Algorithm 1 are described in detail below.

5 Initialization

The following steps of the Algorithm 1 are not critical as far as time is concerned; they are executed just onceCE[b].

5.1 Starting point location

There are several methods for finding a starting point on an implicit surface. These algorithms can be based on some random search methods as in Bloomenthal [5] or

CE[c] Where is Algorithm 2?
CE[d] Dependence on what? Please clarify

on more sophisticated approaches. In Triquet et al. [29], searching in a constant direction from the interior of an implicit object is used.

In our approach, we use a simple algorithm to find a starting point. The starting point is sought in the defined area by following a gradient vector $\nabla \boldsymbol{f}$ of an implicit function f. The algorithm looks for a point $\boldsymbol{p}_0$ that satisfies the equation $f(\boldsymbol{p}_0) = 0$.

5.2 First triangle

The first triangle in polygonization is assumed to lie near a tangent plane of the starting point $\boldsymbol{p}_0$ that is on the implicit surface. Let such point $\boldsymbol{p}_0$ exist. Then the algorithm proceeds as follows.CE[e]

1. Compute the normal vector $\boldsymbol{n} = (n_x, n_y, n_z)$ at the point $\boldsymbol{p}_0$ (see Fig. 1). Note that the normal vector is determined as a unit-length gradient of f; $\boldsymbol{n} = \nabla \boldsymbol{f}/|\nabla \boldsymbol{f}|$.
2. Determine the tangent vector $\boldsymbol{t}$ as in Hartmann [15]. If $(n_x > 0.5)$ or $(n_y > 0.5)$, then $\boldsymbol{t} = (n_y, -n_x, 0)$. Otherwise, $\boldsymbol{t} = (-n_z, 0, n_x)$.
3. Use the tangent vector $\boldsymbol{t}$ as a virtual active edge, and use the edge-spinning algorithm (described below) to compute the coordinates of the second point $\boldsymbol{p}_1$. The points $(\boldsymbol{p}_0, \boldsymbol{p}_1)$ form the first edge e_0.

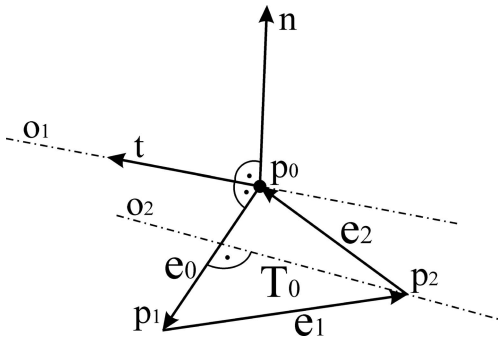

Fig. 1. The first triangle

4. Polygonize the first edge e_0 using the edge-spinning algorithm to get the third point $\boldsymbol{p}_2$. Points $(\boldsymbol{p}_0,\boldsymbol{p}_1,\boldsymbol{p}_2)$ and edges (e_0,e_1,e_2) form the first triangle T_0.

6 Edge-spinning

The main goal of this work is to achieve a numerically stable computation of surface point coordinates for objects defined by implicit functions. Differential properties for each implicit function are different and depend on the modeling technique used [17, 18, 22, 24, 26, 28]; an accurate determination of a surface vertex position depends on them as well. In general, a surface vertex position is searched in the direction of the gradient vector $\nabla \boldsymbol{f}$ of an implicit function f, as in Hartmann [15]. In many cases, the computation of the gradient of the function f is influenced by an error. The well-known marching cubes algorithm uses a stable approach to locate expected surface vertices along the edges of intersecting cells.CE[b]

Because of these reasons, we have defined some restrictions for finding a new surface point $\boldsymbol{p}_{\text{new}}$ as follows:

- The new point $\boldsymbol{p}_{\text{new}}$ is sought at a constant distance from a given edge, i.e., on a circle. The circle radius is proportional to the estimated surface curvature, and therefore, each new generated triangle preserves the desired accuracy of polygonization.
- The circle lies in the plane that is defined by the normal vector of triangle T_{old} and axis o of the current edge e (see Fig. 4); this guarantees that the newly generated triangle is well-shaped (isosceles).

6.1 Determination of the circle radius

The radius of the circle is proportional to the estimated surface curvature.CE[b] The surface curvature radius r_c between points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, with normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$, respectively (see Fig. 2), is estimated by the formula

$$r_c = \frac{d}{\alpha}, \tag{1}$$

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

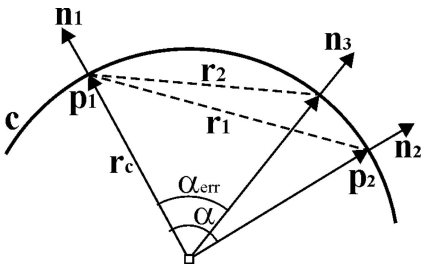

Fig. 2. The circle of surface curvature c with radius r_c between points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, and estimation of radius r_2 of the finding circle according to desired approximation error α_{err}

The radius of surface curvature r_c is evaluated three times among pairs of points $(\boldsymbol{p}_1, \boldsymbol{p}_{init})$, $(\boldsymbol{p}_2, \boldsymbol{p}_{init})$, and $(\boldsymbol{p}_s, \boldsymbol{p}_{init})$, where $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are points of the current active edge e, $\boldsymbol{p}_s$ is the midpoint, and $\boldsymbol{p}_{\text{init}}$ is the point of intersection of the circle c_1 with the plane defined by the triangle T_{old} (see Fig. 3). The final r_c is taken as the minimum of these three.

The new circle radius r_2 is computed as follows:

$$r_2 = k \cdot r_c \cdot \sqrt{2 \cdot (1 - \cos \alpha_{\text{err}})} \tag{2}$$

CE[e] Is this Algorithm 2?

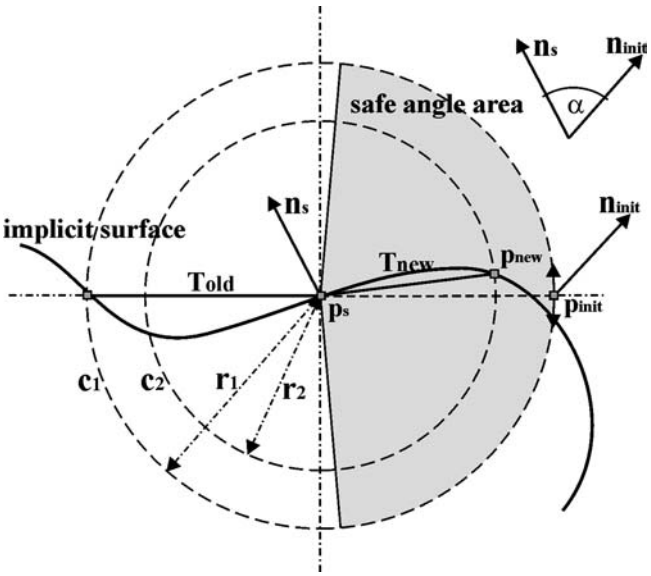

Fig. 3. The finding circle radius estimation

where k is a constant, r_c is the estimated radius of surface curvature, and α_{err} is the required approximation error given at the beginning of the polygonization process.

Note that this formula has been derived from the second cosine theorem $c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos\alpha$ with the presumption $a = b = r_c$ (see Fig. 2). The initial radius r_1 of the circle c_1 is proportional to the length of the current active edge e to a new equilateral triangle T_{new} CE[b]. We experimentally set the $k = 0.8$ just to be sure that the new triangle will satisfy the desired accuracy.

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 given at the beginning of the polygonization process as the maximal required level of detail. Note that the constant 0.1 has been empirically set just to preserve the well-shapedness of the triangles CE[b].

6.2 Root-finding

If the algorithm is given the radius of the circle, the process continues as follows CE[b].

1. Set the point $\boldsymbol{p}_{\text{new}}$ to its initial position; the initial position is in the plane of triangle T_{old} on the other side of edge e (see Fig. 4). Let the angle of the initial position be $\alpha = 0°$.
1. Compute CE[f] the function values $f(\boldsymbol{p}_{\text{new}}) = f(\alpha)$,
   $f(\boldsymbol{p}'_{\text{new}}) = f(\alpha + \Delta\alpha)$ – initial position rotated by the angle $+\Delta\alpha$,
   $f(p''_{\text{new}}) = f(\alpha - \Delta\alpha)$ – initial position rotated by the angle $-\Delta\alpha$;
   Note that the edge e is the rotation axis.

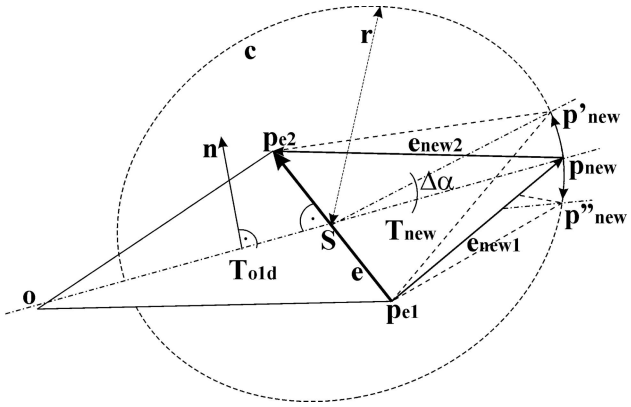

Fig. 4. The principle of the root-finding algorithm

2. Determine the right direction of rotation:
   if $|f(\alpha + \Delta\alpha)| < |f(\alpha)|$, then $+\Delta\alpha$. Otherwise, $-\Delta\alpha$.
3. Let the function values $f_1 = f(\alpha)$ and $f_2 = f(\alpha \pm \Delta\alpha)$; update the angle $\alpha = \alpha \pm \Delta\alpha$.
4. Check which of following cases is satisfied:
   a) If $(f_1 \cdot f_2) < 0$ then compute the accurate coordinates of the new point $\boldsymbol{p}_{\text{new}}$ by via binary subdivision between the last two points corresponding to the function values f_1 and f_2;
   b) If the angle $|\alpha|$ is less than α_{safe} (see "safe angle area" in Fig. 3), return to step 4.
   c) If the angle $|\alpha|$ is greater than α_{safe}, then there is a possibility that both triangles T_{old} and T_{new} could cross each other; the point $\boldsymbol{p}_{\text{new}}$ is rejected, and it is marked as "not found".

Note that in our implementation, we use the limit angle $\alpha_{\text{safe}} = 80°$, which has been determined experimentally.

6.3 Root-finding on a sharp edge

Let us assume that the standard edge-spinning root-finding algorithm presented above has found the point $\boldsymbol{p}_{\text{new}}$. The algorithm then determines the surface normal vector $\boldsymbol{n}_{\text{new}}$ at this point and computes the angle α between normal vectors $\boldsymbol{n}_{\text{new}}$ and $\boldsymbol{n}_s$. The vector $\boldsymbol{n}_s$ is measured at midpoint $\boldsymbol{s}$ of the active edge e (see Fig. 5). If the angle α is greater than a user-specified threshold $\alpha_{\text{lim\_edge}}$ (limit edge angle), then the algorithm will look for a new edge point as follows:

Algorithm 4: Root-finding on a sharp edge. CE[g]

1. Compute coordinates of the point $\boldsymbol{p}_{\text{init}}$ as an intersection of the three planes, tangent planes $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, and the plane in which circle c lies (see Fig. 5).
2. Apply the straight root-finding algorithm and find the new point $\boldsymbol{p}'_{\text{new}}$.

Note that the algorithm needs an accurate determination of surface normal vectors, i.e. accurate com-

CE[f] Is this Algorithm 3?
CE[g] There is no Algorithm 2 or Algorithm 3 labeled
CE[h] Do you mean "continuously differentiable"? "Good differentiable" is not a proper term

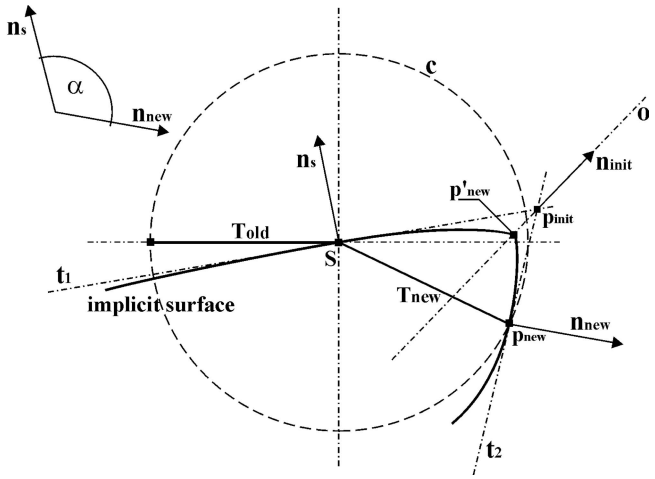

Fig. 5. The principle of the root-finding algorithm for sharp edges

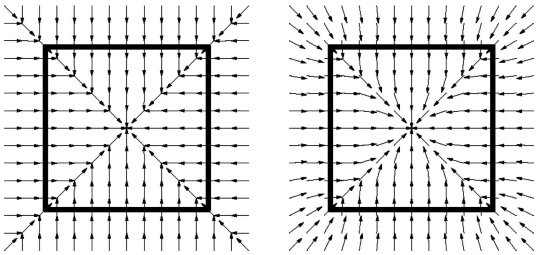

Fig. 6. A square modeled as intersection of four half-spaces, taken from [22]. *Left*: by min/max operations. *Right*: by the F-Rep operations. Arrows represent the direction of a function gradient

putation of a function gradient. Therefore, implicit objects should be modeled by F-Rep [25], because objects defined by the "pure" min/max operations are not good differentiable[CE:h] [22, 26]. Figure 6 shows that the min/max operations create objects with poor differential properties.

6.4 Straight root-finding algorithm

The algorithm starts from an initial point $\boldsymbol{p}_{\text{init}}$ (see Fig. 7) and supposes that the implicit surface is C^0 continuous. The algorithm continues as follows:

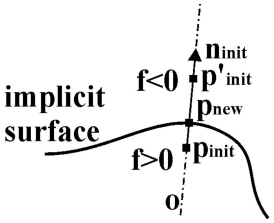

Fig. 7. Principle of root-finding in a straight direction

Algorithm 5: Straight root-finding

1. At point $\boldsymbol{p}_{\text{init}}$, compute the surface normal vector $\boldsymbol{n}_{\text{init}}$ that defines the seeking axis o.
2. Compute coordinates of point $\boldsymbol{p}'_{\text{init}}$ with distance δ from point $\boldsymbol{p}_{\text{init}}$ in direction $\boldsymbol{n}_{\text{init}} * \text{sign}(f(\boldsymbol{p}_{\text{init}}))$, where δ is the length of the step and the function sign returns "1" if $(f > 0)$ or "-1" if $(f < 0)$.
3. Determine the values of functions f and f' at points $\boldsymbol{p}_{\text{init}}$ and $\boldsymbol{p}'_{\text{init}}$.
4. Check which of the next two cases is satisfied.
   a) If these points lie on opposite sides of the implicit surface, i.e., $(f * f') < 0$, compute the exact coordinates of the point $\boldsymbol{p}_{\text{new}}$ by binary subdivision between these points.
   b) If the points $\boldsymbol{p}_{\text{init}}$ and $\boldsymbol{p}'_{\text{init}}$ lie on the same side of the surface, then let $\boldsymbol{p}_{\text{init}} = \boldsymbol{p}'_{\text{init}}$ and return to step 2.

7 Polygonization of an active edge

Polygonization of an active edge e consists of several steps:

Algorithm 6: Active edge polygonization.

1. Use the edge-spinning algorithm to find a new point $\boldsymbol{p}_{\text{new}}$ in front of the edge e.
2. Determine angles α_1 and α_2 in front of points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ of the current edge e (see Fig. 8).
3. Perform a neighborhood test.
4. Perform a distance test.

7.1 Neighborhood test

If the point $\boldsymbol{p}_{\text{new}}$ has been found, there are two cases illustrated in Fig. 8. Deciding between cases (a) and (b) depends on the relation among angles α_1, α_2 and α_n (see Fig. 8). Let the angle α be $\min(\alpha_1, \alpha_2)$. If $(\alpha < \alpha_{shape})$, then we have case (a); otherwise, we have case (b) (see Fig. 8)[CE:i]. The limit shape angle is determined as $\alpha_{shape} = \alpha_n + \pi/6$, so the space for the adjacent triangles should be at least $\pi/6$; this constant just affects the shape of the next generated triangles and has been set experimentally[CE:b].

If the point $\boldsymbol{p}_{\text{new}}$ is not found, angle α_n is not defined, and the limit angle α_{shape} should be less than π; we have experimentally chosen $\alpha_{\text{shape}} = 2/3 * \pi$[CE:b].

1. In this case, a new triangle t_{new} is created by connecting the edge e with one of its neighbors (see step 2(a)).
2. The new triangle t_{new} is created by joining the active edge e and the new point $\boldsymbol{p}_{\text{new}}$ (see step 2(b)).

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.

[CE:i] Or do you mean " If $(\alpha < \alpha_{shape})$, then invoke case (a); otherwise, invoke case (b)"?

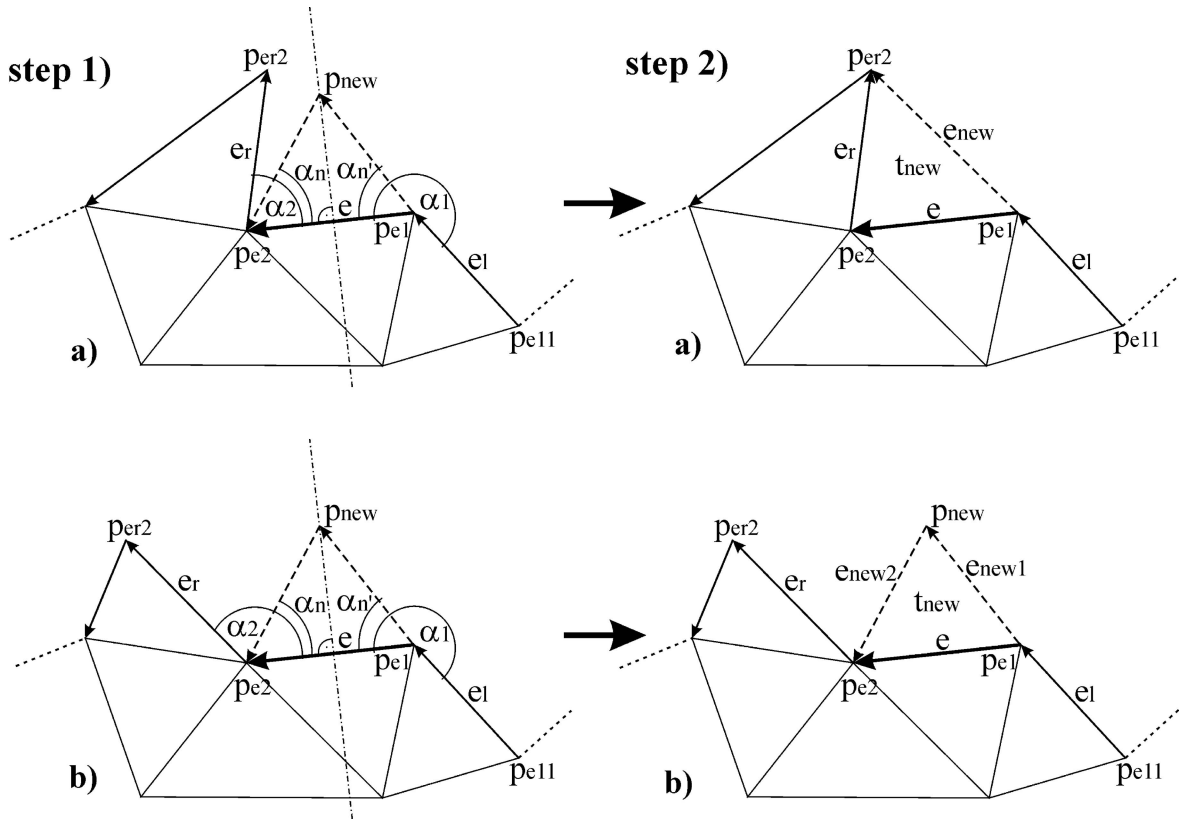

Fig. 8. Polygonization of the active edge e

Note that if there is no new triangle (the point $\boldsymbol{p}_{\text{new}}$ does not exist and case (a) has not been satisfied) the bounding sphere of the active edge e is used. The next procedure is analogous for all cases.

7.2 Distance test

To preserve the correct topology, it is necessary to check each newly generated triangle to see if it does not cross a surface already generated. The distance test should be performed between the new triangle and a border of an already triangulated area (i.e., the active edges in the AEL).

The algorithm will generate a "nearest active edge list" (NAEL) for the new triangle t_{new}. Each active edge that is not adjacent to the current active edge e and crosses the bounding sphere of the new triangle (or edge e), is included on the list (see Fig. 9, step 2). The extended bounding sphere is used for the new triangle created by the new point $\boldsymbol{p}_{\text{new}}$ (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 have experimentally chosen the constant $c = 1.3$.

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

In case (a) in Fig. 8, 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 the section below.

In case (b) in Fig. 8, step 2, the current active edge e is deleted from the list and the new edges, e_{new1} and e_{new2}, are added at the end of the list.

Note that if there is no new triangle to be created (the point $\boldsymbol{p}_{\text{new}}$ does not exist and case (a) in Fig. 8 has not been satisfied) the current active edge e is moved to the end of the AEL, and the entire Algorithm 1 returns to step 2.

If the NAEL is not empty, then the situation has to be solved. The point $\boldsymbol{p}_{\text{min}}$ with the minimal distance from the current edge e is chosen from the NAEL (see Fig. 9, step 3).

This point has to satisfy a condition of thin objects as well. The current active edge e and the point $\boldsymbol{p}_{\text{min}}$ should not lie on opposite sides of the implicit surface. Figure 10 illustrates this unfeasible situation.

If the correct point $\boldsymbol{p}_{\text{min}}$ is found, the new triangle t_{new} has to be changed and will be formed by the edge e and the point $\boldsymbol{p}_{\text{min}}$, i.e., by points ($\boldsymbol{p}_{e1}$, $\boldsymbol{p}_{\text{min}}$ and $\boldsymbol{p}_{e2}$); the situ-

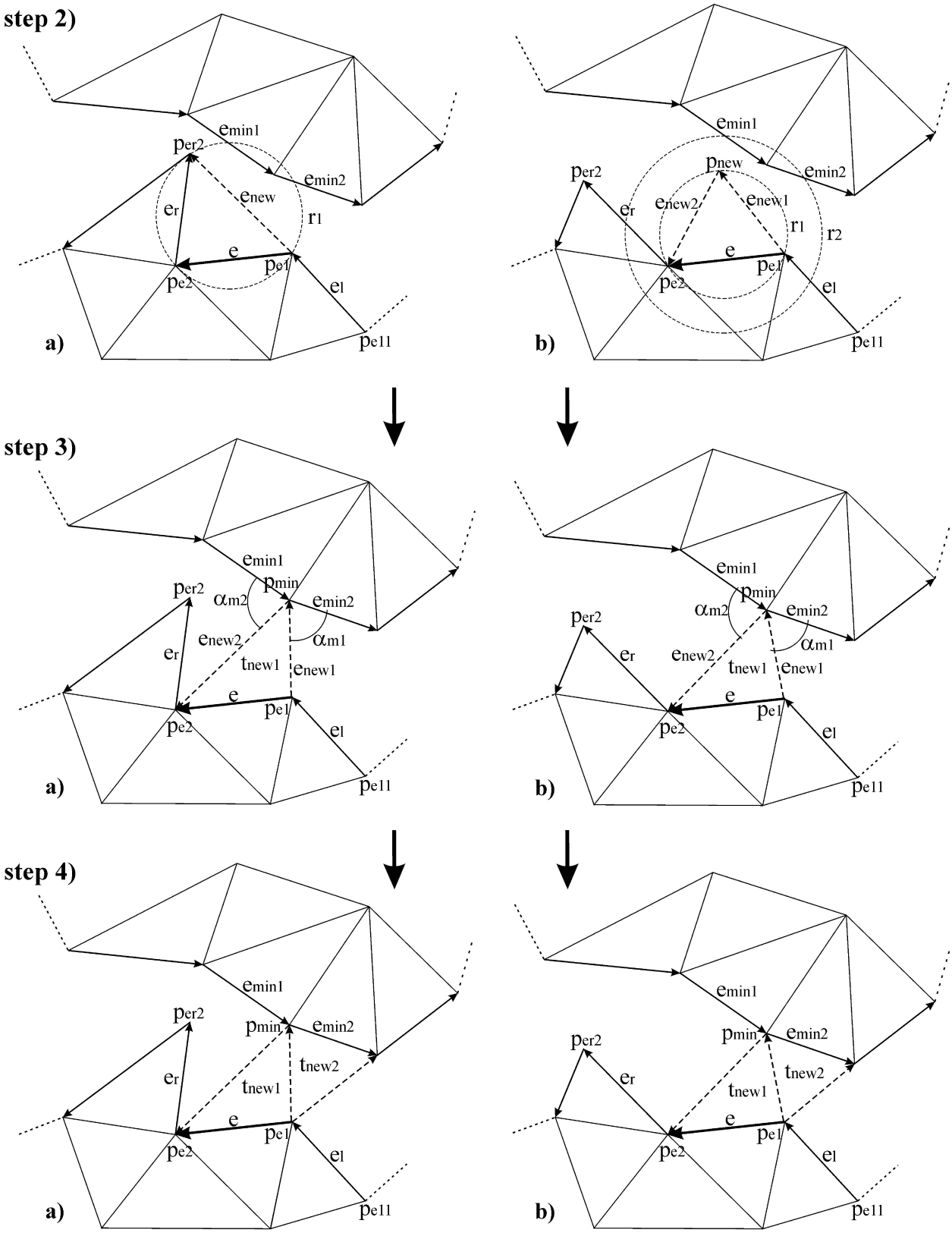

Fig. 9. Solving of the distance test

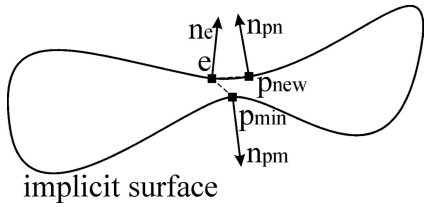

Fig. 10. A problem of thin implicit objects

ation is described in Fig. 9, step 3. The point $\boldsymbol{p}_{\min}$ belongs to four active edges – e_{new1}, e_{new2}, e_{min1} and e_{min2} – and the border of the already triangulated area intersects itself at that point[CE b]. This is not correct because each point that lies on the triangulation border should have only two neighborhood edges (left and right).

The solution of this problem is to triangulate two of the four edges first. Let the four active edges be divided into pairs; the left pair is (e_{min1}, e_{new2}) and the right pair is (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} and α_{m2} (see Fig. 9, step 3). If ($\alpha_{m1} < \alpha_{m2}$), then the left pair is polygonized; otherwise, 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 $\boldsymbol{p}_{\min}$ is contained only in one pair of active edges, and the border of the triangulated area is correct (see Fig. 9, step 4).

Note that the polygonization of one pair of edges simply consists of joining its endpoints by the edge. This second new triangle has to fulfill the empty NAEL as well; otherwise, the current active edge e is moved to the end of the AEL.

7.3 Splitting the new triangle

This process is evaluated only in cases when the new triangle has been created by connecting two adjacent edges, i.e., the situation illustrated in Fig. 8, step 2(a). If the new edge does not comply with the condition of a surface curvature, the new triangle should be split into two new triangles (see Fig. 11, step 2). This occurs when the angle α between surface normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ at points $\boldsymbol{p}_{e1}$ and $\boldsymbol{p}_{er2}$ is greater than the given limit α_{err} (see Fig. 11 step 1)[CE b]. The triangle will also be split in the case when the connecting edge is longer than the limit value $\text{LOD}_{\max}$ given at the beginning.

The point $\boldsymbol{p}_{\text{new}}$ is a midpoint of edge e_{new}, and it does not lie on the implicit surface. Its correct coordinates are computed by the straight root-finding algorithm.

8 Polygonization of sharp objects

Our previously developed method [9, 10] has been used only to polygonize implicit surfaces which comply with C^1 continuity or have only simple sharp edges; i.e. it cannot be used with corners or more complicated shapes[CE b]. In order to solve this problem, we used a simple method that supposes that an object is modeled by the F-Rep [17]. It allows us to assume that an implicit function has sharp edges only in its zero-set, i.e., points $\boldsymbol{x}_i$ that satisfy equation $f(\boldsymbol{x}_i) = 0$. If the algorithm will look for some isovalue ε, the equation will change to $f(\boldsymbol{x}_i) = \varepsilon$, and the implicit surface is then C^1 continuous (see Fig. 12).

The discussed implicit surface property allows us to construct an initial mesh that satisfies a desired accuracy in relation 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 $\boldsymbol{x}_i$ on the original implicit surface, i.e., $\varepsilon = 0$, $f(\boldsymbol{x}_i) = 0$. There are various algorithms that work with an initial mesh and iteratively adapt it to get more precise approximations [20, 22, 23]. In order to verify our approach, we propose a simple algorithm. The points $\boldsymbol{x}_i$ follow their gradient $\nabla \boldsymbol{f}(\boldsymbol{x}_i)$ to find the new positions. This method is similar to particle system approaches [14], but it follows an opposite order of

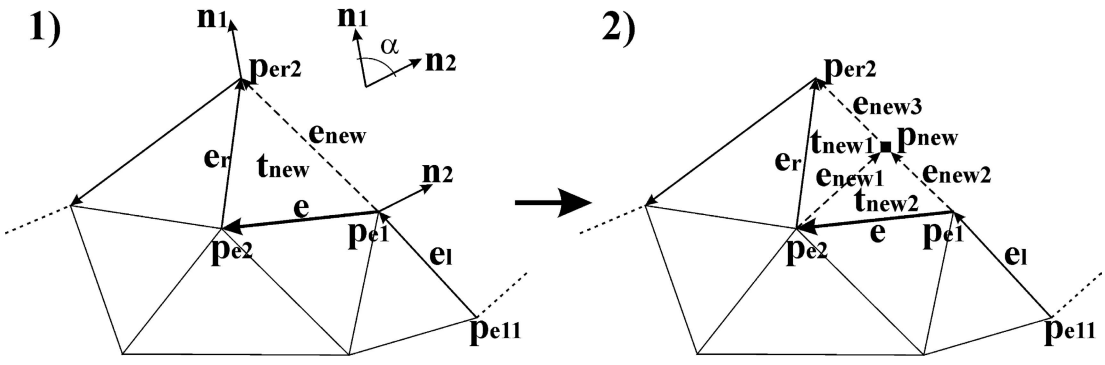

Fig. 11. Splitting of the new triangle

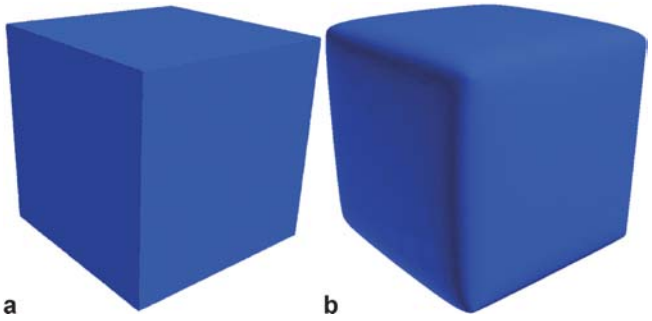

Fig. 12. A cube modeled by F-Rep as the intersection of six half-spaces and polygonized with (a) $\varepsilon = 0$ – classical approach, function is C^0 continuous, and (b) $\varepsilon = 0.1$, function is C^1 continuous

steps; the triangulation is created first, and then the points are projected to the implicit surface.

The algorithm is similar to the straight root-finding algorithm (see Algorithm 5) with the difference being that the surface normal vector is computed in each step.

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

1. Set $\varepsilon = 0$.
2. For each point $\boldsymbol{x}_i$, compute its new normal vector $\boldsymbol{n}_i$.
3. Move the point to its new position $\boldsymbol{x}'_i$ in the normal vector direction,
   $\boldsymbol{x}'_i = \boldsymbol{x}_i + \delta * \text{sign}(f_i)\, \boldsymbol{n}_i$,
   where δ is a step and sign(f_i) is the signum function of the function value f_i at the point $\boldsymbol{x}_i$. Note that in our approach, we set the step size $\delta = 1/3 * \leq_{\min}$, where $\leq_{\min}$ is the minimal length of a triangle edge such that the value is proportional to the variable $r_{\min}$ described in Sect. 6.1. This value influences the speed of convergence.
4. Determine the values of functions f_i and f'_i at points $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$.
5. Check which of the next two cases is satisfied .
   a) If these points lie on opposite sides of an implicit surface, i.e., $(f_i * f'_i) < 0$; compute the exact coordinates of the point $\boldsymbol{x}_i$ by binary subdivision between these points.
   b) If the points $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$ lie on the same side of the surface, then let $\boldsymbol{x}_i = \boldsymbol{x}'_i$ and return to step 2.

9 Experimental results

The adaptive edge-spinning algorithm (AES) is based on the surface-tracking scheme (also known as the continuation scheme). Therefore, we have compared it with other methods based on the same principle – the marching triangles algorithm (MTR), introduced in Hartmann [15] and the marching cubes method (MC), Bloomenthal's polygonizer (see [5]).

As the first testing function, the implicit object Genus 3 has been chosen; it is defined as follows, taken from Hartmann [15].

$$f(\boldsymbol{x}) = r_z^4 \cdot z^2 - \left[1 - (x/r_x)^2 - (y/r_y)^2\right] \cdot \left[(x - x_1)^2 + y^2 - r_1^2\right] \cdot \left[(x + x_1)^2 + y^2 - r_1^2\right] = 0 \tag{3}$$

where the parameters are set as follows: $r_x = 6$, $r_y = 3.5$, $r_z = 4$, $r_1 = 1.2$, and $x_1 = 3.9$.

The measured values from the experiment are shown in Table 1. 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 the size of a cube cell.

Table 1 contains the number of triangles and vertices generated. The value *Angle err* is proportional to surface curvature and denotes the average deviation between surface normal vectors at points sharing an edge. For the edge-spinning algorithm, it corresponds to α_{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.

The values *Alg dist avg*, *Euc dist avg*, and *Taub dist avg* measure the approximation quality as an average distance of a triangle from the real implicit surface. They are measured at the gravity centre of each triangle. The distance is either algebraic (*Alg dist avg*), real Euclidian (*Euc dist avg*), or the Taubian (*Taub dist avg*) [28].

Note that the algebraic distance (function value) strongly depends on the given implicit function and it is only proportional to the real distance. The Euclidian distance has been measured between a triangle centroid and

Table 1. Values of the object Genus 3 measured with variable level of detail (LOD)

GENUS 3	Edge-spinning	Marching triangles	Marching cubes
LOD	4.00×10^{-2}	4.50×10^{-2}	4.57×10^{-2}
Triangles	331 832	310 811	334 816
Vertices	165 912	155 401	167 404
Angle err	7.96×10^{-3}	8.23×10^{-3}	8.12×10^{-3}
Centroid angle err	1.89×10^{-3}	1.94×10^{-3}	2.29×10^{-3}
Alg dist avg	1.62×10^{-1}	1.72×10^{-1}	2.03×10^{-1}
Euc dist avg	1.09×10^{-4}	1.15×10^{-4}	1.40×10^{-4}
Taub dist avg	1.09×10^{-4}	1.15×10^{-4}	1.40×10^{-4}
Angle criterion	0.853	0.719	0.368
Edge length criterion	0.908	0.821	0.526
Time (ms)	7801	4267	2564
Time avg (ms)	23.509	13.729	7.658

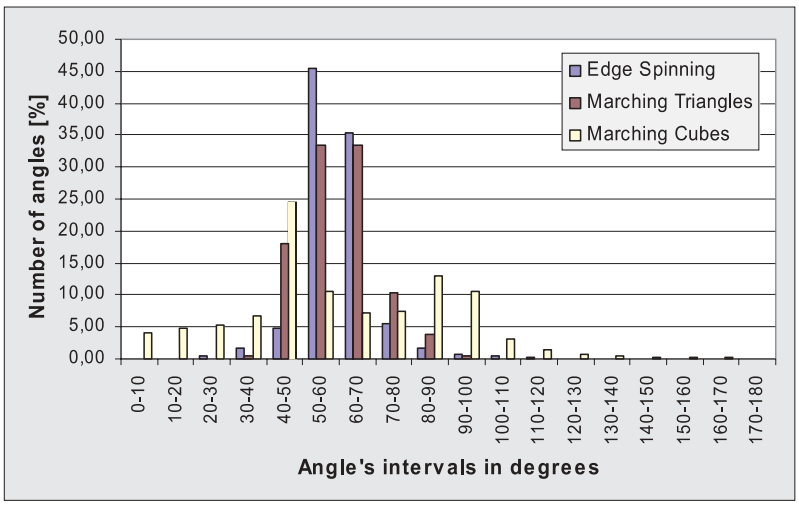

Fig. 13. Histogram of triangle shape quality for the edge-spinning, the marching triangles and the marching cubes algorithms, generated according to values in Table 1

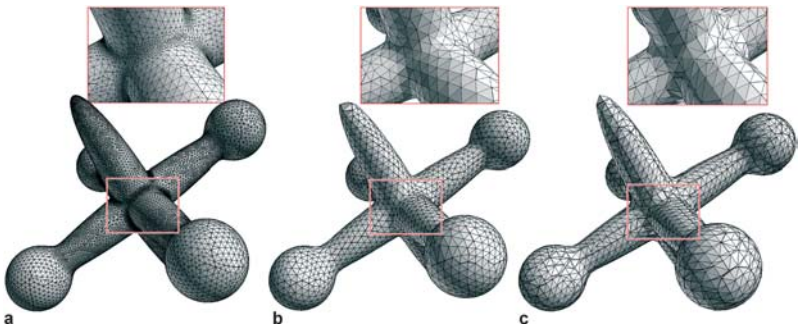

Fig. 14. The jack object generated by the (a) edge-spinning algorithm, (b) marching triangles algoritm, and (c) marching cubes algorithm. Above each jack is a zoomed-in image for better illustration of the details

its corresponding surface point by applying the straight root-finding algorithm (Algorithm 5) to the triangle centroid to find the surface point. For the function Genus 3, the Taubian distance is a good approximation to the real distance.

The *Angle criterion* denotes the ratio of the smallest angle to the largest angle in a triangle, and the *Edge length criterion* denotes the ratio of the shortest edge to the longest edge of a triangle$_{CE}^{b}$. These values show the quality of resulting triangles generated.

The histogram in Fig. 13 illustrates the triangulation quality as well. It can be seen that the edge-spinning method generates about 80% triangles with angles in the interval $< 50, 70 >$ degrees.

The value *Time* in Table 1 shows the measured computational time of each algorithm, and *Time avg* represents the average time needed to create 1000 triangles.

Note that the time values are included only for a better illustration of the algorithm properties because the presented method is primarily focused on the quality of approximation, not on the speed. In our test, the accelerated version of the MTR method [12] has been used and its results correspond to the method$_{CE}^{b}$.

Another test is aimed at comparing the surface approximation quality with the same maximum LOD values for all polygonization methods. The test is performed on the Jack object, introduced in Bloomenthal[5] and shows advantages over the adaptive approach.

The adaptive edge-spinning algorithm shrinks the size of triangles in regions of higher curvature; therefore, the number of triangles is greater then the number generated by the other non-adaptive algorithms. The precision of the

Table 2. Values of the jack object measured with a constant level of detail for all methods

JACK	Edge-spinning	Marching triangles	Marching cubes
LOD	0.16	0.16	0.16
Triangles	34 256	6107	6552
Vertices	17 130	3055	3278
Angle err	3.33×10^{-2}	7.47×10^{-2}	7.54×10^{-2}
Centroid angle err	7.10×10^{-3}	1.40×10^{-2}	2.13×10^{-2}
Alg dist avg	2.39×10^{-3}	1.32×10^{-2}	1.67×10^{-2}
Euc dist avg	8.29×10^{-4}	4.62×10^{-3}	5.93×10^{-3}
Taub dist avg	8.30×10^{-4}	4.66×10^{-3}	5.97×10^{-3}
Angle criterion	0.700	0.729	0.377
Edge length criterion	0.806	0.828	0.536
Time (ms)	1442	70	71
Time avg (ms)	42.095	11.462	10.836

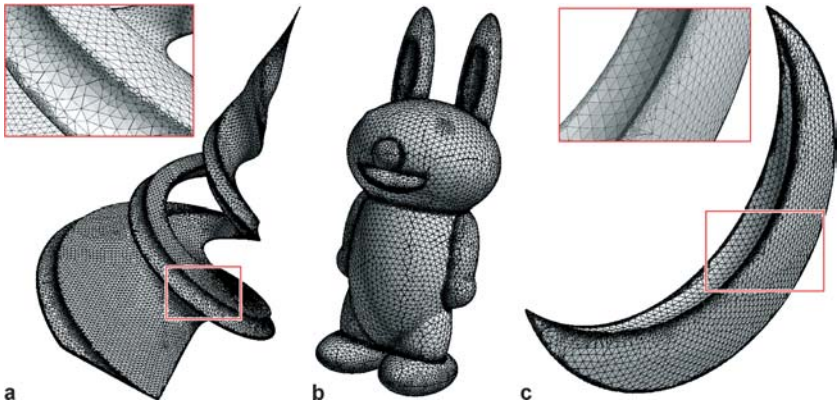

Fig. 15. Objects generated by the adaptive edge-spinning algorithm using the introduced technique. (a) An object taken from [22]. (b) The rabbit modeled by F-Rep [17]. c) The eclipse model

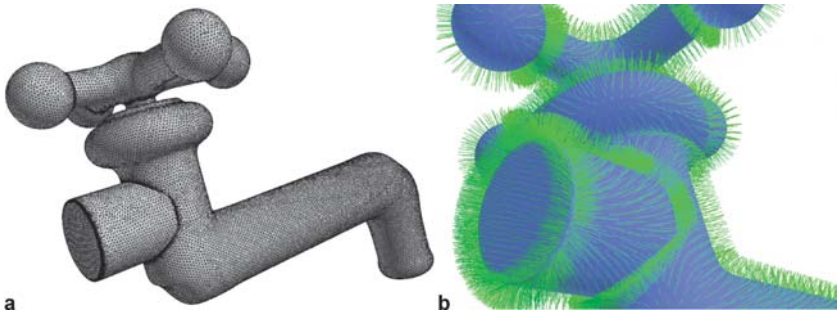

Fig. 16. A tap generated by the adaptive edge-spinning algorithm using the introduced method. (a) Its triangulation. (b) The array of normal vectors

polygonization is higher by about one order of magnitude as well (see Table 2).

The algorithm can also polygonize objects with sharp features. Triangulations of some examples are shown in Fig. 15.

Fig. 16 shows the implicit model of a tap with its normal vector array. The tap object contains sharp edges as well.

Note that the tap and the rabbit are modeled with use of an implicit modeling module [30], which is a part of the modular visualization environment (MVE) developed at University of West Bohemia [21].

Table 3 contains values measured on complex implicit objects visualized in the figures mentioned above. There, you can see that the Taubian distance is not a good enough approximation of the real distance for such complex objects.

The average time values to create 1000 triangles are higher in comparison with the simpler models because the one call of the function takes more time.

Table 3. Values generated by the edge-spinning algorithm

Edge-spinning	Yutaka	Rabbit	Eclipse	Tap
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	5.31×10^{-2}	4.55×10^{-2}	6.94×10^{-2}	4.40×10^{-2}
Centroid angle err	1.43×10^{-2}	1.06×10^{-2}	3.15×10^{-2}	1.00×10^{-2}
Alg dist avg	9.76×10^{-2}	9.79×10^{-2}	9.32×10^{-1}	9.37×10^{-2}
Euc dist avg	1.42×10^{-3}	2.06×10^{-3}	1.11×10^{-3}	3.03×10^{-3}
Taub dist avg	8.85×10^{-2}	1.77×10^{-1}	2.29×10^{-2}	6.10×10^{-2}
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	7841	6519	7741
Time avg (ms)	91.524	161.573	208.721	202.728

10 Conclusion

This paper presents a new adaptive approach for polygonization of implicit surfaces. The algorithm marches over the object's surface and computes accurate coordinates of new points by spinning the edges of already generated triangles. The resulting triangulation is fully adapted to the surface curvature estimation. As a local curvature estimation, we used the deviation of angles of adjacent points because it is simple and fast for computation. According to this estimation, the new points are sought on a circle with radius computed to satisfy precisely the desired error. Therefore, the whole algorithm preserves the required level of detail and the approximation error given at the beginning of the process.

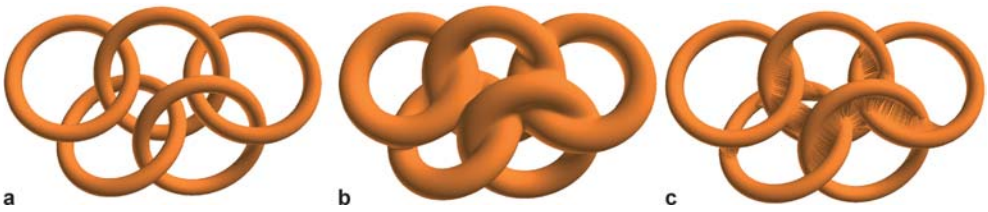

Fig. 17. (a) the original "olympic rings" object. (b) Olympic rings polygonized with $\varepsilon = 10$. (c) Visualization after projection back to $\varepsilon = 0$

The main advantage of our algorithm, in comparison with other methods [1, 18], is the aforementioned control of approximation quality during computation. The whole process is directed to achieve the given accuracy, and the algorithm maintains this requirement in all places of an implicit object (with high or low curvature). This means that the resulting polygonal mesh not only consists of well-shaped triangles, but the mesh satisfies predefined requirements of accuracy as well.

A similar curvature estimation has been used in the literature [3, 18, 31], and our experiments proved the functionality of the method as well.

The algorithm can polygonize a variety of implicit surfaces whose size and degree of continuity is not known beforehand. It is possible to use the introduced technique when there is a non-zero ϵ value of an implicit function rather than a zero value[CE:b].

Of course there must be limitations to this feature. There is no exact way to predict the value of ϵ. It depends on the size of the object, the sharpness of edges, etc. Moreover, for higher values of ϵ, an implicit object could change its topology (see Fig. 17). In such cases, the projection phase cannot work properly.

In our future research, we would like to propose an algorithm that will make possible the polygonization of unknown complex implicit scenes that consists of more disjoint implicit objects. The algorithm will use our introduced method for each object separately, but it should be possible to use the algorithm with other surface approaches as well. We are currently work on such an algorithm.

Acknowledgement The authors of this paper would like to thank to all who contributed to development of this approach, especially to the MSc and PhD students at the University of West Bohemia in Plzen[CE:j].

References

1. Akkouche S, Galin E (2001) Adaptive implicit surface polygonization using marching triangles. Comput Graph Forum 20(2):67–80
2. Allgower EL, Gnutzmann S (1987) An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces. SIAM J Numer Anal 24:452–469
3. Alliez P, Cohen-Steiner D, Devillers O, Lévy B, Desbrun M (2003) Anisotropic polygonal remeshing. In: SIGGRAPH 2003, ACM TOG 22(3):485–493
4. Baumgart's winged-edge http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/winged-e.html[CE:k]
5. Bloomenthal J (1994) Graphics gems IV. Academic Press, New York
6. Bloomenthal J (1995) Skeletal design of natural forms. Dissertation,[CE:l]
7. Bloomenthal J, Bajaj Ch, Blinn J, Cani-Gascuel M-P, Rockwood A, Wyvill B, Wyvill G (1997) Introduction to implicit surfaces. Kaufmann, San Francisco
8. Bloomenthal J (1988) Polygonization of implicit surfaces. Comput-Aided Geom Des 5(4):341–355
9. Čermák M, Skala V (2004) Adaptive edge spinning algorithm for polygonization of implicit surfaces. In: Proceedings of Computer Graphics International, CGI 2004, Crete, Greece[CE:m]
10. Čermák M, Skala V (2002) Polygonization by the edge spinning. In: Proceedings of the International Conference Algoritmy 2002, Slovakia[CE:m]
11. Čermák M, Skala V (2002) Accelerated edge spinning algorithm for implicit surfaces. In: Proceedings of the International Conference ICCVG 2002, Zakopane, Poland[CE:m]
12. Čermák M, Skala V (2002) Space subdivision for fast polygonization of implicit surfaces. In: Proceedings of the International Conference ECI 2002, Slovakia[CE:m]
13. Figueiredo LH (1992) Computational morphology of implicit curves. Dissertation, IMPA[CE:n]
14. Figueiredo LH, Gomes JM, Terzopoulos D, Velho L (1992) Physically-based methods for polygonization of implicit surfaces. In: Proceedings of Graphics Interface 92:250–257
15. Hartmann E (1998) A marching method for the triangulation of surfaces. Vis Comput 14:95–108
16. Hilton A, Stoddart AJ, Illingworth J, Windeatt T (1996) Marching triangles: range image fusion for complex object modelling. In: Proceedings of the International Conference on Image Processing 2:381–384
17. "Hyperfun: language for F-Rep geometric modeling," http://cis.k.hosei.ac.jp/~F-rep/[CE:o]
18. Ju T, Losasso F, Schaefer S, Warren J (2002) Dual contouring of Hermite data. In: SIGGRAPH 2002, pp 339–346
19. Karkanis T, Stewart AJ (2001) Curvature-dependent triangulation of implicit surfaces. IEEE Comput Graph Appl 21(2):60–69
20. Kobbelt LP, Botsch M, Schwanecke U, Seidel H-P (2001) Feature sensitive surface extraction from volume data. In: SIGGRAPH 2001 proceedings, pp 57–66
21. MVE – modular visualization environment project, http://herakles.zcu.cz/research.php, 2001[CE:o]
22. Ohtake Y, Belyaev A, Pasko A (2003) Dynamic mesh optimization for polygonized implicit surfaces with sharp features, Vis Comput 19:115–126
23. Ohtake Y, Belyaev AG (2002) Dual/primal mesh optimization for polygonized implicit surfaces. In: Proceedings of the ACM Solid Modeling Symposium, Saarbrucken, Germany, pp 171–178
24. Pasko A, Adzhiev V, Karakov M, Savchenko V (2000) Hybrid system architecture for volume modeling. Comput Graph 24:67–68
25. Rvachov AM [CE:p] Definition of R-functions, http://www.mit.edu/~maratr/rvachev/p1.htm
26. Schmidt MFW (1993) Cutting cubes – visualizing implicit surfaces by adaptive polygonization. Vis Comput 10(2):10–115

[CE:j] This is spelled "Pilsen" in your contact information. Please [CE:k] choose one or the other
Can you provide additional details for this reference, e.g. author name, date, etc.?
[CE:l]
Please supply the institution at which this dissertation was

27. Shapiro V, Tsukanov I (1999) Implicit functions with guaranteed differential properties. In: Proceedings of the Fifth Annual Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, pp 258–269
28. Taubin G (1994) Distance approximations for rasterizing implicit curves. ACM Trans Graph 13:3–42
29. Triquet F, Meseure F, Chaillou C (2001) Fast polygonization of implicit surfaces. In: Proceedings of the International Conference WSCG, pp 283–290
30. Uhlir K, Skala V (2003) The implicit function modelling system – comparison of C++ and C# solutions. In: Proceedings of C# and NET Technologies' 2003, University of West Bohemia, Czech Republic
31. Velho L (1996) Simple and efficient polygonization of implicit surfaces. J Graph Tools 1(2):5–24
32. Velho L, Figueiredo L, Gomes J (1999) A unified approach for hierarchical adaptive tessellation of surfaces. ACM Trans Graph 18(4):329–360

MARTIN ČERMAK is a member of Computer Graphics Group at the University of West Bohemia in Plzen. He has received his PhD at the Department of Computer Science and Engineering.

VÁCLAV SKALA is a full professor at the University of West Bohemia in Plzen. He is responsible for the Centre of Computer Graphics and Visualization (http://herakles.zcu.cz) and he is the Head of Computer Graphics Group at University of West Bohemia in Pilsen.