Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Applied Numerical Mathematics 49 (2004) 331–342
APPLIED NUMERICAL MATHEMATICS
www.elsevier.com/locate/apnum

Edge spinning algorithm for implicit surfaces ☆

M. Cermak, V. Skala *

Department of Computer Science, University of West Bohemia, Univerzitni 8, Box 314, 306 14 Plzen, Czech Republic

Abstract

This paper presents a new fast method for polygonization of implicit surfaces. Our method put emphasis on the shape of triangles generated and on the polygonization speed. The main advantages of the triangulation presented are simplicity and the stable features that can be used for future expansion. The implementation is not complicated and only the standard data structures are used. This algorithm is based on the surface tracking scheme and it is compared with the well-known marching cubes algorithm that is based on the similar principle. The presented algorithm is accelerated by the space subdivision which is an effective technique to speed-up any geometric of this type.

Keywords: Polygonization; Triangulation; Implicit surfaces; Marching method; Edge spinning; Acceleration

1. Introduction

Implicit surfaces seem to be one of the most appealing concepts for 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 in space **x** that satisfy the equation $f(\mathbf{x}) = 0$. Thus, visualizing implicit surfaces typically consists in 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 for implicit surfaces. The first one [2,3] defines an implicit object as $f(\mathbf{x}) < 0$ (function $f_1(\mathbf{x})$ below) and the second, F-rep [7,12], (functional representation, function $f_2(\mathbf{x})$) defines it as $f(\mathbf{x}) \geqslant 0$. In our implementation, we use the F-rep definition of implicit objects. The implicit functions described below show the differences between both definitions for the function sphere.

$$f_1(\mathbf{x})\colon\ x^2 + y^2 + z^2 - r^2 = 0, \qquad f_2(\mathbf{x})\colon\ r^2 - x^2 - z^2 = 0.$$

☆ This work was supported by the Ministry of Education of the Czech Republic – project MSM 235200005.
* Corresponding author.
E-mail addresses: cermakm@kiv.zcu.cz (M. Cermak), skala@kiv.zcu.cz (V. Skala).

0168-9274/$30.00
doi:10.1016/j.apnum.2003.12.011

Iso-surface extraction is needed for the visualization purposes that have a set of triangles as a result. Existing techniques may be classified into three categories.

Spatial sampling techniques regularly or adaptively sample the space to find the cells straddling the implicit surface, and tessellate those cells to create overall polygonization [2,3,9]. In general, cells are either cubes or tetrahedra.

Surface tracking approaches (also known as continuation method) iteratively create a triangulation from a seed element by marching along the surface [1,2,5,6,14].

Surface fitting techniques progressively adapt and deform an initial mesh to converge to the implicit surface.

2. Data structures

The presented algorithm uses only the standard data structures used in computer graphics. The main data structure is the edge that is used as a basic building block for polygonization. We use the standard winding edge and therefore, the resulting polygonal mesh is correct and complete with neighborhood among all triangles generated. The basic data structures used there are:

- edge—winding edge;
- active edge—an edge that lies on the triangulated area's border; implemented as an index into winding edge's array;
- list of active edges—dynamically allocated list of active edges;
- point—if a point lies on an active edge it contains also two pointers to left and right active edge; left and right directions are in active edges orientation.

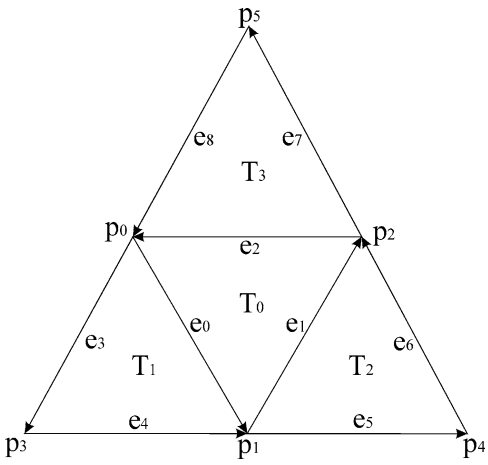

Fig. 1. The first steps of the algorithm.

3. Principle of our algorithm

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

(1) Find a starting point $\mathbf{p}_0$.
(2) Create a first triangle T_0, see Fig. 1.
(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 at the end of the active edges list.
(6) Check the distance between the new generated point $\mathbf{p}_{\text{new}}$ and all the other points which lie on the border of already triangulated area (which lie in all the other active edges).
(7) If the active edges list is not empty return to step 4.

4. 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 [2] or on more sophisticated approach. In [14], 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 a defined area in the direction of a gradient vector ∇f of an implicit function f. The algorithm looks for a point $\mathbf{p}_0$ that satisfies the equation $f(\mathbf{p}_0) = 0$.

5. 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.

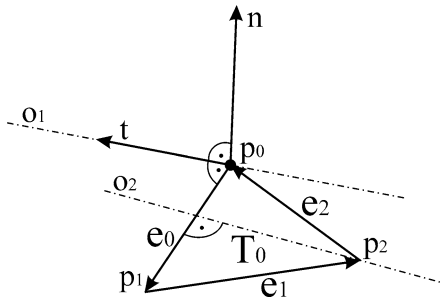

Fig. 2. The first triangle generation.

(1) Determine the normal vector $\mathbf{n} = (n_x, n_y, n_z)$ in the starting point $\mathbf{p}_0$, see Fig. 2,

$$\mathbf{n} = \nabla\mathbf{f}/\|\nabla\mathbf{f}\|.$$

(2) Determine the tangent vector $\mathbf{t}$ as in [5]. 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 $\mathbf{t}$ as a fictive active edge and use the edge spinning algorithm (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.

6. Edge spinning algorithm

The main goal of this work is a numerical stability of a surface point coordinates' computation for objects which are defined by the implicit function. Differential properties for each implicit function are different in dependence on the modeling techniques [6,7,10,12,13] and the accurate determination of a position of a surface vertex depends on them. In general, a surface vertex position is searched in direction of a gradient vector of an implicit function f, e.g., in [5]. In many cases, the computing of a gradient of the 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 $\mathbf{p}_{\text{new}}$.

- The new point $\mathbf{p}_{\text{new}}$ is sought in a constant distance, i.e., on a circle; then each new generated triangle preserves the desired accuracy of polygonization—the average edge's length δ_e. The circle radius is proportional to the δ_e.
- The circle lies in the plane that is defined by the normal vector of triangle T_{old} (see Fig. 3) and axis o of the actual edge e; this guarantees that the new generated triangle is well shaped (isosceles).

Then, the algorithm is:

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

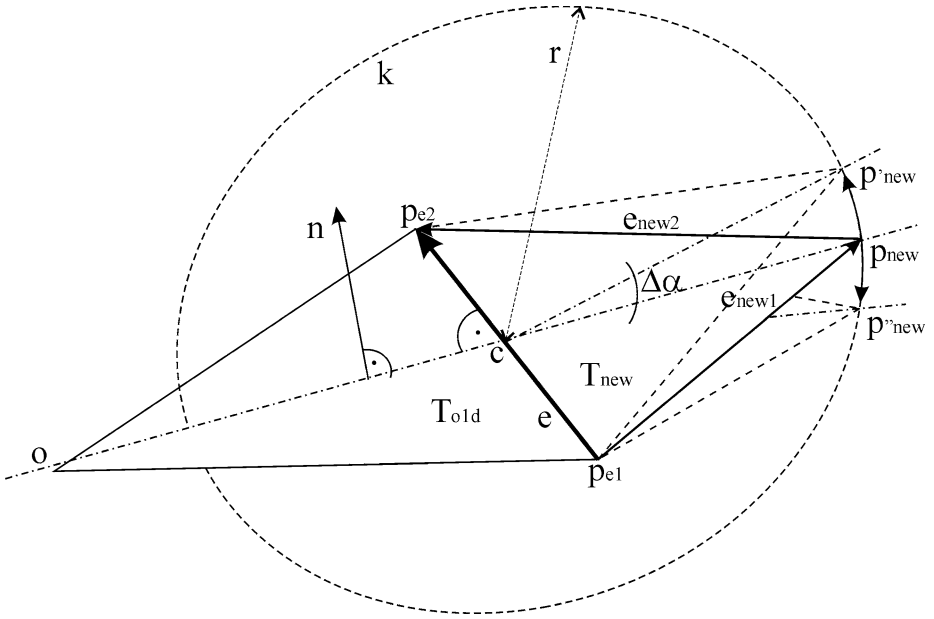

Fig. 3. The edge spinning algorithm principle.

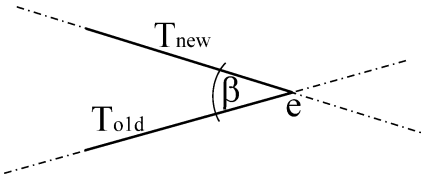

Fig. 4. The angle between two triangles; the view is in direction of edge's vector e.

7. 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 case appeared, see Fig. 5.

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

The relations among limit angles are $\alpha_{\text{lim\_1}} < \alpha_{\text{lim\_2}} \leqslant \alpha_{\text{lim\_3}}$.
Possible cases which are illustrated in Fig. 5 are:

(a) In this case, algorithm creates a new triangle and includes a new active edge e_{new} to the end of the active edges list.

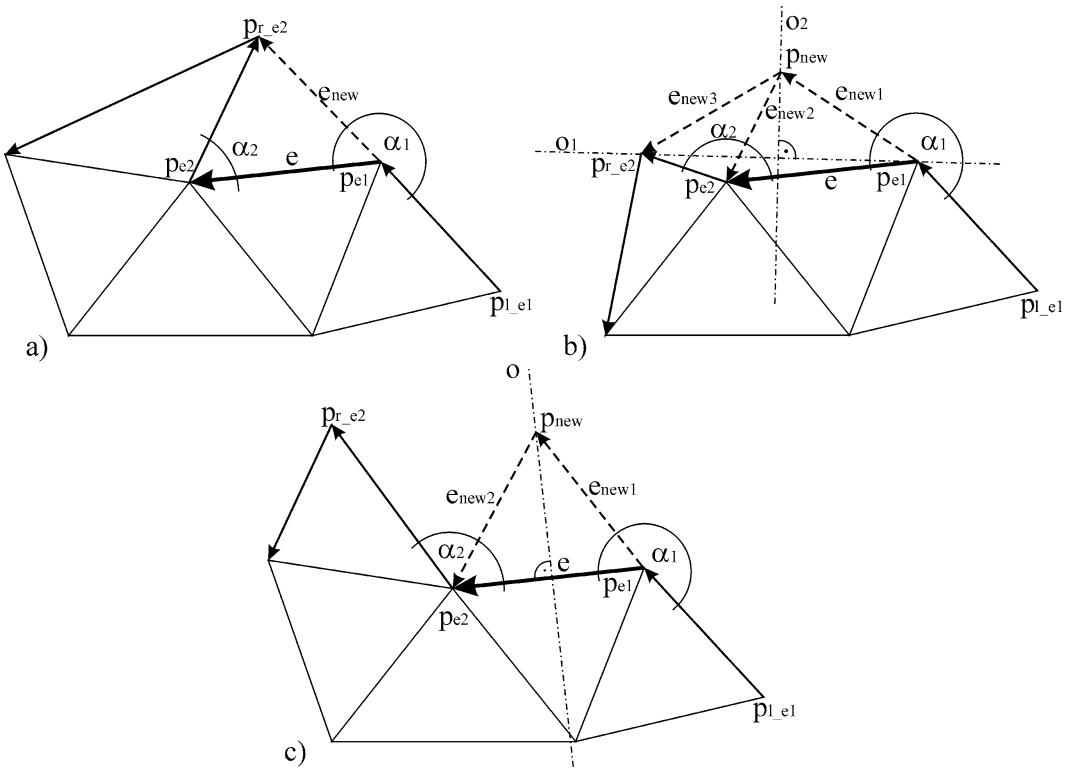

Fig. 5. The possible cases for polygonization of an active edge e.

(b) In some situations, the length of certain edges can be shorter then the 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 Fig. 5) is used as a fictive active edge for the algorithm edge spinning and the new point $\mathbf{p}_{\text{new}}$ is created as well as two new triangles.

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

8. 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 $\mathbf{p}_{\text{new}}$ must be checked for distance with all the other points which lie in active edges (on the triangulation border).

Let the point $\mathbf{p}_{\text{min}}$ be the nearest point to this new point $\mathbf{p}_{\text{new}}$ and distance between both points is $\delta = \|\mathbf{p}_{\text{new}} - \mathbf{p}_{\text{min}}\|$. Further, let $\mathbf{p}_{\text{min}}$ not lie in the active edges which are in the neighborhood of both active edges which contain the point $\mathbf{p}_{\text{new}}$. Then there are two cases described in Fig. 6.

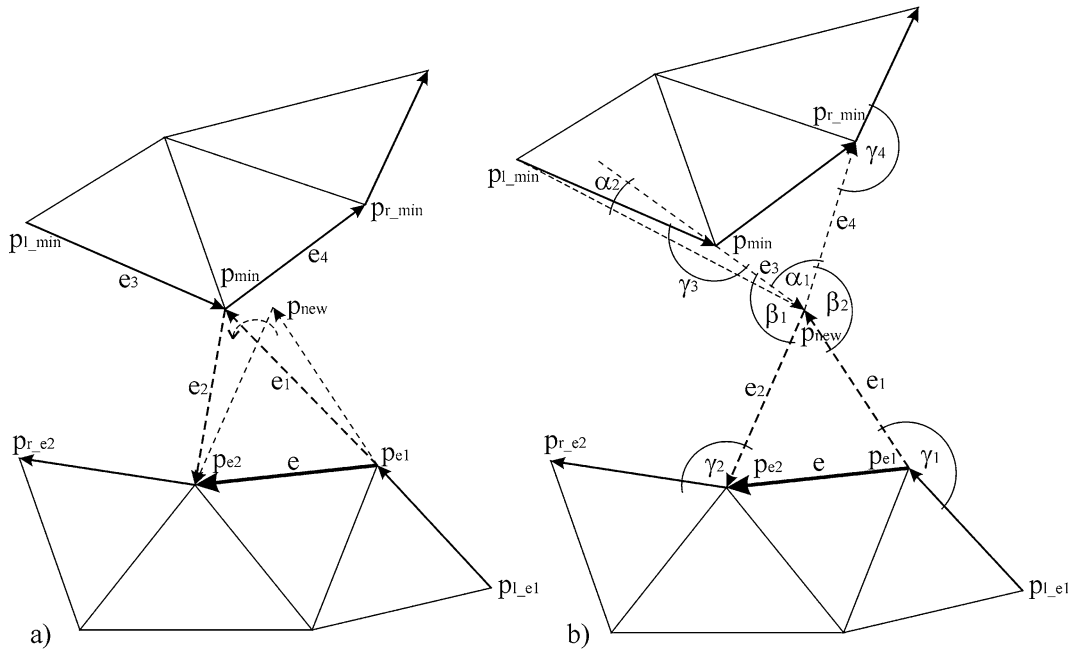

Fig. 6. The possible cases for distance test.

(a) If $\delta < \delta_{\text{lim\_3}}$ then the new point $\mathbf{p}_{\text{new}}$ is replaced with the point $\mathbf{p}_{\text{min}}$.
(b) If $\delta < \delta_{\text{lim\_4}}$ then a new triangle must be created between the new point $\mathbf{p}_{\text{new}}$ and one of two active edges which contain the point $\mathbf{p}_{\text{min}}$, i.e., either the triangle $(\mathbf{p}_{\text{min}}, \mathbf{p}_{\text{new}}, \mathbf{p}_{r\_\text{min}})$ or the triangle $(\mathbf{p}_{l\_\text{min}}, \mathbf{p}_{\text{new}}, \mathbf{p}_{\text{min}})$, see Fig. 6(b). 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 $\langle 0, \pi \rangle$ and therefore, the triangle with the angle α_i that is better approximation of angle 90° is chosen.

The relation between distance limits is $\delta_{\text{lim\_3}} < \delta_{\text{lim\_4}}$.

Now the situation described in Fig. 6(a) and (b) is similar for both cases. Point $\mathbf{p}_{\text{new}}$ 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. The solution of the problem will be introduced in case (b) and solution for case (a) is analogical. 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 Fig. 6(b). 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 $\mathbf{p}_{\text{new}}$ is contained only in one pair of active edges.

In Fig. 6(b), 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 as the first; else the active edge e_2 is polygonized first. Now, the border of the triangulated area does not cross itself in the point $\mathbf{p}_{\text{new}}$ 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.

9. Space subdivision principle

The original distance check algorithm takes more time if a required scene detail grows (growing number of points on the triangulation border). The algorithm complexity for one included point is $O(N)$, where N is a number of points in active edges. In aspect 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 nearest point ahead.

Advantageous solution is dividing of space into sub-spaces (sub-areas), similarly as in [4]. Then the nearest point must lie in the same sub-area like the new included point or in the closest neighborhood. In order to validate this theorem next equation must be valid as well: $\sigma \geqslant \delta_{\text{lim}\_4}$, where σ is the size of sub-areas (cube shape), see Fig. 7, and δ_{lim_4} is the high limit distance for distance check.

The algorithm complexity of this solution is $O(M)$, where M is a number of points in adjacent sub-areas and $M \ll N$. Fig. 7 shows 9 possible sub-areas in E^2 case and there are 27 possible sub-areas in E^3 case.

In implementation, the data structures mentioned in Section 2 must be slightly expanded.

- sub-areas—an array; each sub-area has its own dynamically allocated list of points which lie in;
- point–each point that lies on the triangulation border has the index of sub-area in which the point lies as well.

The computational time of the Edge spinning algorithm consists of two main parts, the *polygonization time* and the *distance test time*. Fig. 8 shows the ratio between both parts for the edge spinning algorithm without and with usage of the space subdivision technique.

It is obvious that such acceleration technique is effective in result and simple for implementation. More examples and tests are found in next section.

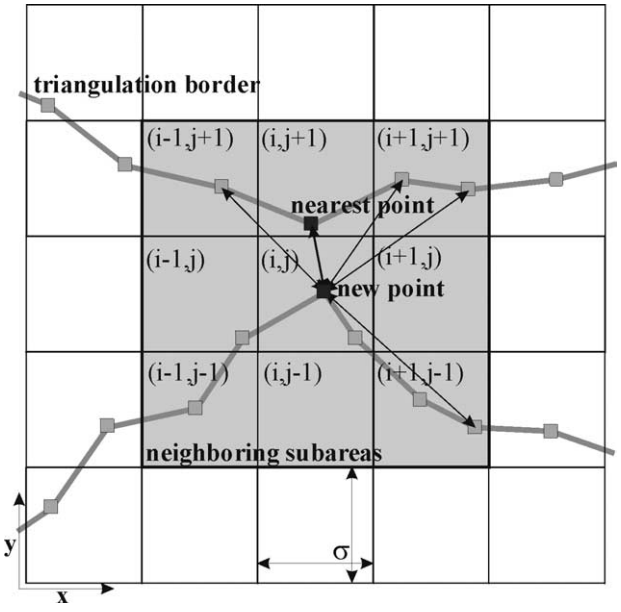

Fig. 7. The space subdivision scheme for distance checking.

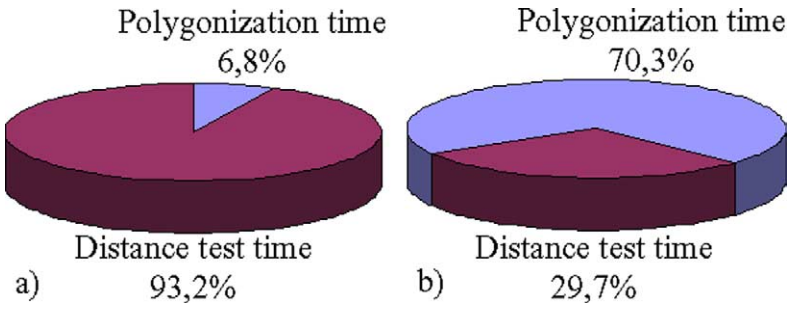

Fig. 8. The time ratio between the polygonization time and the distance test time of the edge spinning algorithm; (a) without the space subdivision scheme, (b) with the space subdivision.

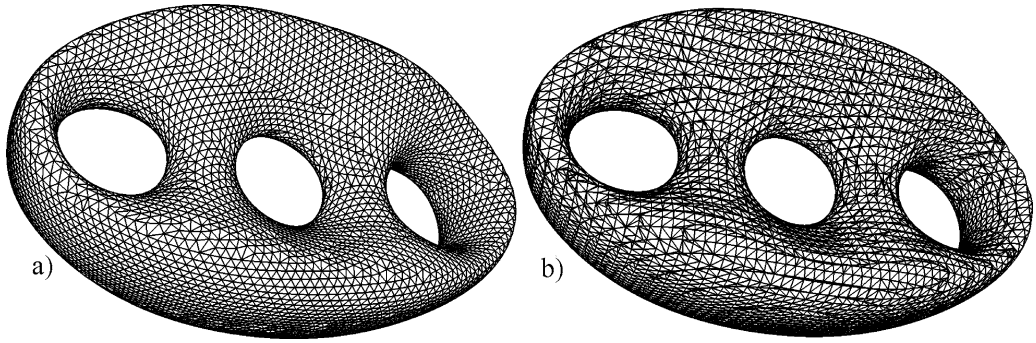

Fig. 9. The triangular mesh of the implicit object Genus that is polygonized by (a) the edge spinning algorithm, (b) the marching cubes algorithm.

10. Experimental results

All the experiments described above were accomplished on the implicit object *Genus*, see Fig. 9. Its implicit function is described as follows:

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

where $\mathbf{x} = [x, y, z]$ and the parameters are: $r_x = 6$, $r_y = 3.5$, $r_z = 4$, $r_1 = 1.2$, $x_1 = 3.9$.

A visual comparison of a triangle's shape quality proves that the polygonal mesh generated by the edge spinning algorithm is much better than the marching cubes method. This result is vindicated by the next histogram, Fig. 10 that shows the percentage frequency of triangles' angles generated in the output polygonal mesh.

The original edge spinning (ES) algorithm (without usage the space subdivision technique) takes more time for polygonization of implicit objects than the marching cubes (MC) algorithm. This deficiency is successfully remedied in the accelerated version. Fig. 11 shows the polygonization time's ratios between mentioned algorithms. This diagram proves that the accelerated edge spinning (ESA) algorithm is stably (constantly) faster than the marching cubes algorithm in all experiments. Fig. 11 contains the speed-up between both versions of the edge spinning algorithm as well and demonstrates the efficiency of the acceleration technique used.

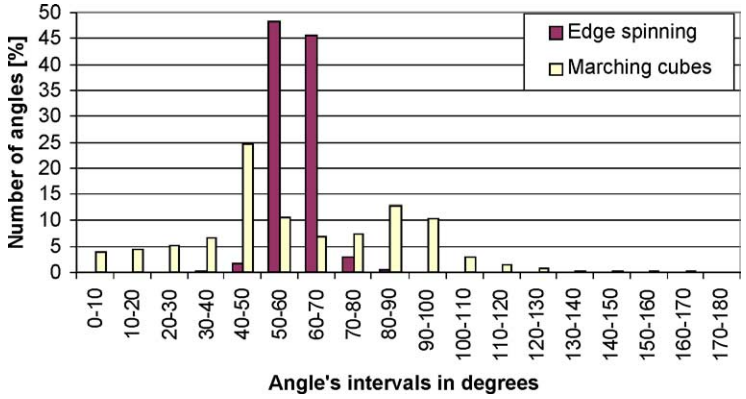

Fig. 10. Histogram of the triangle shape quality for the edge spinning and the marching cubes algorithms. Generated for $N = 1000$, see Table 1.

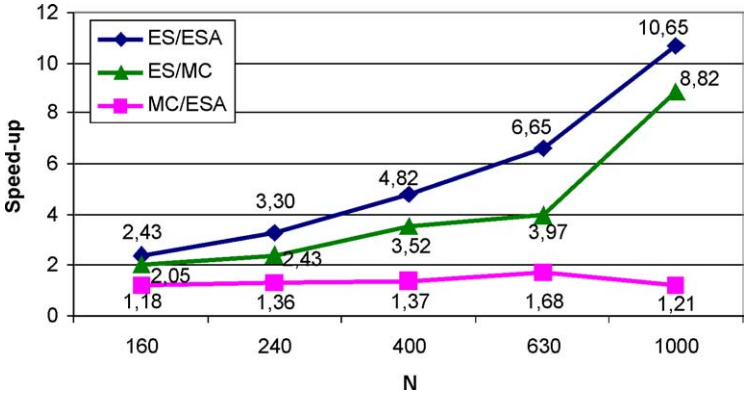

Fig. 11. Computational time comparison (speed-up) between the original ES algorithm and the accelerated one, between the original ES and the MC algorithm and between the ESA and the MC method.

The N variable used in figures above and in Table 1, represents a desired object detail. Specifically, the average triangles edges' length is proportional to N and to the computing area's size as well. With growing N, the triangles' edges are shorter, i.e., each side of the computing area is like divided into N parts and length of such part is the average edges' length of generated triangles. The computing area's size used in experiments is $[\langle x_{\min}, x_{\max}\rangle, \langle y_{\min}, y_{\max}\rangle, \langle z_{\min}, z_{\max}\rangle] = [\langle -8, 8\rangle, \langle -8, 8\rangle, \langle -8, 8\rangle]$.

The measured values from all experiments described above are contained in Table 1.

It is obvious that the edge spinning algorithm generates about 23% triangles less than the marching cubes method and the output polygonal mesh consists of well-shaped triangles. The results presented are original and also were verified on many nontrivial implicit surfaces.

11. Conclusion

In this paper, we have presented the new principle for polygonization of implicit surfaces. The algorithm marches over the object's surface and computes the accurate coordinates of new points

Table 1
The measured values from experiments

	N	160	240	400	630	1000
Edge spinning	Triangles:	53992	120766	335622	834966	2110538
	Vertices:	26992	60379	167807	417479	1055265
	Time [ms]:	380	871	2664	7511	21161
Marching cubes	Triangles:	69676	157520	437800	1085376	2735836
	Vertices:	34826	78756	218896	542684	1367914
	Time [ms]:	450	1182	3646	12588	25546

by spinning the edges of already generated triangles. The algorithm can be simply accelerated and experimental results prove that the selection of subset of candidate points by the space subdivision scheme is an effective way for this type of geometric algorithms. The edge spinning algorithm generates a well-shaped triangular mesh and its polygonization speed is comparable with the well-known and simple marching cubes algorithm.

Presented method can polygonize implicit surfaces which comply C^1 continuity. In future work, we want to modify the current algorithm for implicit functions with only C^0 continuity. We suppose that our defined restrictions, for polygonization of an active edge, are the right way. In next research, we will work on adapting the Edge spinning algorithm to local curvature of an implicit surface, [1,8].

Acknowledgements

The authors of this paper would like to thank all those who contributed to development of this new approach, especially to colleagues MSc. and PhD. students at the University of West Bohemia in Plzen. The project presented was implemented as a part of the MVE (Modular Visualization Environment), [7,11].

References

[1] S. Akkouche, E. Galin, Adaptive implicit surface polygonization using marching triangles, Comput. Graph. Forum 20 (2) (2001) 67–80.
[2] J. Bloomenthal, Graphics Gems IV, Academic Press, New York, 1994.
[3] J. Bloomenthal, Skeletal Design of Natural Forms, Ph.D. Thesis, University of Calgary, Calgary, 1995.
[4] M. Cermak, V. Skala, Space subdivision for fast polygonization of implicit surfaces, in: 5th Int. Conf. ECI 2002, Slovakia, October 10–11, 2002, ISBN 80-7099-879-2.
[5] E. Hartmann, A marching method for the triangulation of surfaces, Visual Comput. 14 (1998) 95–108.
[6] A. Hilton, A.J. Stoddart, J. Illingworth, T. Windeatt, Marching triangles: range image fusion for complex object modelling, in: Internat. Conf. Image Processing, Lausanne, Switzerland, IEEE, Piscataway, NJ, 1996.
[7] Hyperfun: Language for F-Rep Geometric Modeling, http://cis.k.hosei.ac.jp/~F-rep/.
[8] MVE—Modular Visualization Environment project, http://herakles.zcu.cz/research.php, University of West Bohemia in Plzen, Czech Republic, 2001.
[9] Y. Ohraje, A. Belyaev, A. Pasko, Dynamic meshes for accurate polygonization of implicit surfaces with sharp features, in: Shape Modeling International, Genua, Italy, IEEE, Piscataway, NJ, 2001, pp. 74–81.
[10] A. Pasko, V. Adzhiev, M. Karakov, V. Savchenko, Hybrid system architecture for volume modeling, Comput. Graphics 24 (2000) 67–68.

[11] M. Rousal, V. Skala, Modular visualization environment—MVE, in: Int. Conf. ECI 2000, Herlany, Slovakia, 2000, pp. 245–250, ISBN 80-88922-25-9.
[12] A.M. Rvachov, Definition of R-functions, http://www.mit.edu/~maratr/rvachev/p1.htm.
[13] V. Shapiro, I. Tsukanov, Implicit functions with guaranteed differential properties, ACM Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, ACM Press, New York, 1999, pp. 258–269.
[14] F. Triquet, F. Meseure, Ch. Chaillou, Fast polygonization of implicit surfaces, in: WSCG 2001 Int. Conf., University of West Bohemia in Pilsen, 2001, p. 162.