Computer Graphics International 2004 conference proccedings, IEEE, ISSN 1530-1052, 2004

Adaptive Edge Spinning Algorithm for Polygonization of Implicit Surfaces

Martin Čermák[1] and Václav Skala[2]
University of West Bohemia, Pilsen
Department of Computer Science and Engineering
Czech Republic
{cermakm|skala}@kiv.zcu.cz

Abstract

This paper presents an adaptive method for polygonization of implicit surfaces. The method insists on the shape of triangles and the accuracy of resulting approximation as well. The main advantages of the triangulation presented are simplicity and the stable features that can be used for next expanding. The implementation is not complicated and only the standard data structures are used. The presented algorithm is based on the surface tracking scheme and it is compared with the other algorithms based on the similar principle, such as the Marching cubes and the Marching triangles algorithms.

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 of implicit surfaces. The first one [4], [5] defines an implicit object as $f(\mathbf{x}) < 0$ and the second one, F-rep [14], [17], [18], defines it as $f(x) \geq 0$. In our implementation, we use the F-rep definition of implicit objects. If *f* is an arbitrary procedural method (i.e. a 'black box' function that evaluates **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 **x** and the surface. The measure is Euclidean if it is 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) that regularly or adaptively sample the space to find the cells that straddle the implicit surface [2], [4], [6]. Surface tracking approaches (also known as continuation methods) iteratively create a triangulation from a seed element by marching along the surface [1], [4], [7], [12], [15], [22]. Surface fitting techniques progressively adapt and deform an initial mesh to converge to the implicit surface, [17]. 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, [10], [11]. The desired polygonal approximation is then obtained by computing the Delaunay triangulation associated with the points.

2. Data structures

The presented algorithm uses only the standard data structures used in computer graphics. The main data structure is a winding edge that is used as a basic building block for polygonization.

If a triangle's edge lies on the triangulation border, it is contained in the *active edges list* (*AEL*) and it is called as an *active edge*. Each point, which is contained in an active edge, contains two pointers to its left and right active edge (left and right directions are in active edges' orientation)

For faster evaluation of detection of global overlap (see section 6.2), we use the space subdivision acceleration technique, introduced in [8]. Therefore,

This work was supported by the Ministry of Education of the Czech Republic – projects [1]MSM 235200002 and [2]LA 240

each point in *AEL* also contains the index of sub-area in which it lies. Each sub-area has its own dynamically allocated list of points located inside. The sub-areas are implemented as an array and each of them has its unique index.

3. Principle 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:

1. Initialize the polygonization:
   a. Find the starting point $\mathbf{p}_0$ 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, see Figure 1.
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.

4. Initializing the polygonization

4.1. 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 [4] or on more sophisticated approach. In [22], 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 in a defined area by following of a gradient vector $\nabla\mathbf{f}$ of an implicit function *f*. The algorithm looks for a point $\mathbf{p}_0$ that satisfies the equation $f(\mathbf{p}_0) = 0$.

4.2. The 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. Let such point $\mathbf{p}_0$ exists then the algorithm is as follows.

1. Determine the normal vector $\mathbf{n} = (n_x,n_y,n_z)$ in the starting point $\mathbf{p}_0$, see Figure 1.
2. Determine the tangent vector $\mathbf{t}$ as in [12]. 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 below) 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.

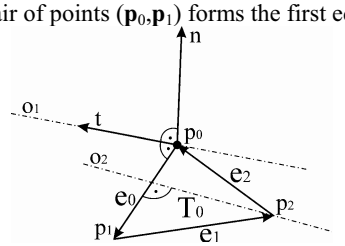

Figure 1. The first triangle.

4. Polygonize the first edge e_0 by 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.

5. Edge spinning

The main goal of this work is a numerical stability of a surface point coordinates' computation for objects defined by implicit functions. Differential properties for each implicit function are different in dependence on the modeling techniques [14], [15], [17], [18], [20], [21] 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 $\nabla\mathbf{f}$ of an implicit function *f*, as in [12]. In many cases, the computation of 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 $\mathbf{p}_{new}$:

- The new point $\mathbf{p}_{new}$ is sought in a constant distance, i.e. on a circle; then each new generated triangle preserves the desired accuracy of polygonization. The circle radius is proportional to the estimated surface curvature.
- 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 Figure 3; this guarantees that the new generated triangle is well shaped (isosceles).

5.1. Determination of the circle radius

The circle radius is proportional to the estimated surface curvature. The surface curvature in front of current active edge is determined in according to angle α between the surface normals $\mathbf{n}_1$, $\mathbf{n}_2$, see Figure 2.

The normal vector $\mathbf{n}_1$ is computed at point **s** that lies in the middle of the current active edge *e* and the vector $\mathbf{n}_2$ is taken at initial point $\mathbf{p}_{init}$ that is a point of intersection of the circle c_1 with the plane defined by the triangle T_{old}.

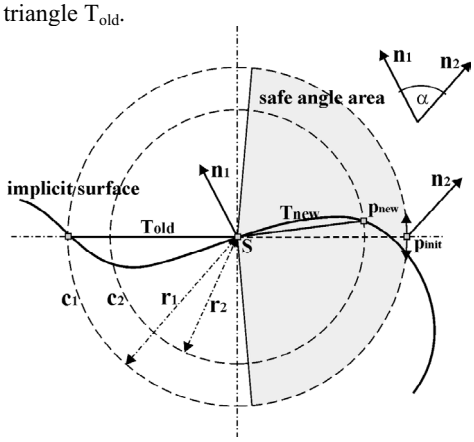

Figure 2. The circle radius estimation.

Note that the initial radius r_1 of the circle c_1 is always the same and it is set at beginning of polygonization as the lowest desired level of detail (LOD). The new circle radius r_2 is computed as follows.

$$r_2 = r_1 \cdot k, k \in \langle 0,1 \rangle;$$

$$k = \left(\frac{\alpha_{\lim} - \alpha \cdot c}{\alpha_{\lim}} \right),$$

where α_{lim} is a limit angle and the constant *c* represents a speed of "shrinking" of the radius according to the angle α. To preserve well shaped triangles, we use a constant k_{min} that represents a minimal multiplier. In our implementation we used $\alpha_{min} = \pi/2$, $k_{min} = 0.2$ and $c = 1.2$.

Correction notes:

if ($\alpha > \alpha_{min}$) then $k = k_{min}$

if ($k < k_{min}$) then $k = k_{min}$

These parameters affect a shape of triangles of the polygonal mesh generated.

5.2. Root finding

If the algorithm knows the circle radius, the process continues as follows.

1. Set the point $\mathbf{p}_{new}$ to its initial position; the initial position is on the triangle's T_{old} plane on the other side of the edge *e*, see Figure 3. Let the angle of the initial position be α=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 $+\Delta\alpha$, $f(p''_{new}) = f(\alpha - \Delta\alpha)$ - initial position rotated by the angle $-\Delta\alpha$; Note that 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 $-\Delta\alpha$.

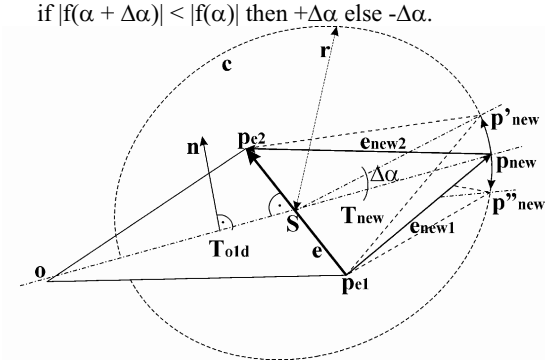

Figure 3. The principle of root finding algorithm.

4. Let the function values $f_1 = f(\alpha)$ and $f_2 = f(\alpha \pm \Delta\alpha)$; update the angle $\alpha = \alpha \pm \Delta\alpha$.
5. Check which of following case appeared:
   a) 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 which correspond to the function values f_1 and f_2;
   b) If the angle $|\alpha|$ is less than α_{safe} (see *safe angle area* in Figure 2) 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 $\mathbf{p}_{new}$ is rejected and it is marked as *not found.*

5.3. Root finding of 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 4. 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.

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 4.
2. Apply the straight root finding algorithm described in section 5.4 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.

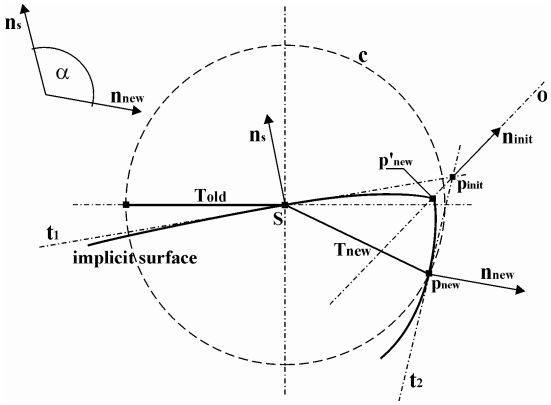

Figure 4. The principle of root finding algorithm for sharp edges.

Therefore, implicit objects should be modeled by F-Rep, [19], because objects defined by min/max operations are not good differentiable, [17], [20].

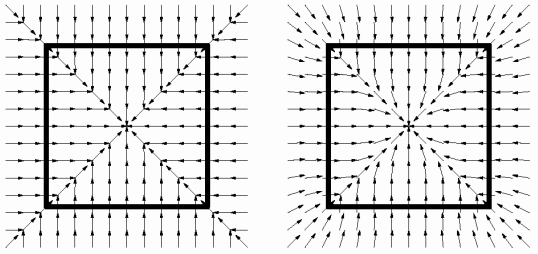

Figure 5. A square modeled as intersection of four half-spaces; left: by min/max operations; right: by the F-Rep operations; taken from [17].

The gradient array of a *square*, modeled by min/max and F-Rep operations, is illustrated in Figure 5. The picture shows that the min/max operations create objects with poor differential properties.

5.4. Straight root finding algorithm

The algorithm starts from an initial point $\mathbf{p}_{init}$ (see Figure 6) and supposes that the implicit surface is at least C^0 continuity.

The algorithm continues as follows.

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 length of step and the function sign returns "1" if ($f > 0$) or "0" if ($f < 0$).
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.

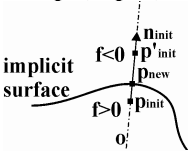

Figure 6. Principle of root-finding in straight direction.

6. Polygonization of an active edge

Polygonization of an active edge *e* consists of several steps. In step 1, the process will use the root finding algorithm (see section 5.2) to find a new point $\mathbf{p}_{new}$ in front of the edge *e*. If $\mathbf{p}_{new}$ exists, there are two cases illustrated in Figure 7.

6.1. Neighborhood test

Decision between cases a) and b) depends on relation among angles α_1, α_2, α_n, see Figure 7; let the angle α be $min(\alpha_1,\alpha_2)$.

If ($\alpha < \alpha_{shape}$) then case a) else case b), see Figure 7; The limit shape angle is determined as $\alpha_{shape} = k*\alpha_n$, $k \geq 1$, $\alpha_{shape} < \pi$, where the constant *k* has effect to shape of generated triangles and in our implementation is chosen $k = 1.7$. If the point $\mathbf{p}_{new}$ is not found, angle α_n is not defined and the limit shape angle should be just less then π; we have chosen $\alpha_{shape} = \pi*0.8$.

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.

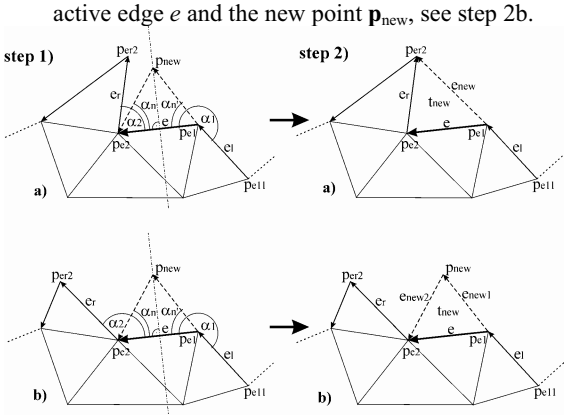

Figure 7. Polygonization of the active edge *e*.

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.

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.

6.2. Distance test

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

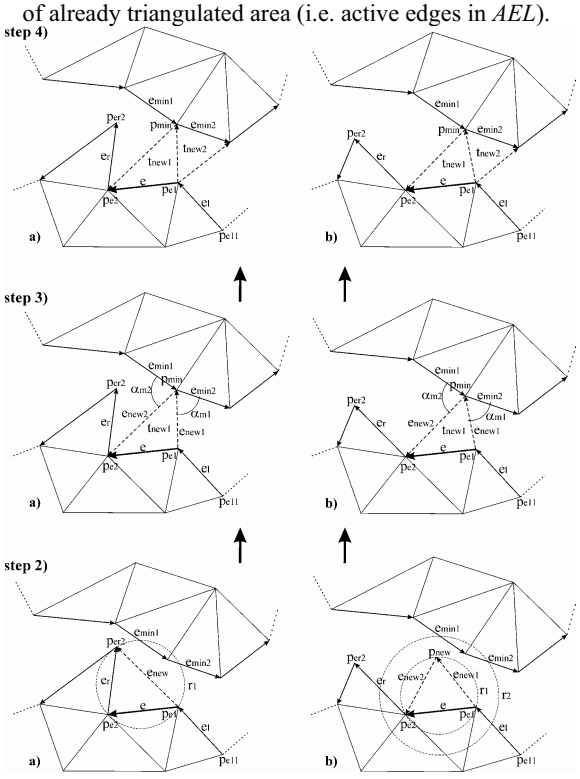

Figure 8. Solving of distance test.

The algorithm will make the *nearest active edges list* (*NAEL*) to 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 the edge *e*), is included to the list, see Figure 8, 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 = 1.5.

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 7 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 6.3.

In case b), Figure 7 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 7 has not appeared) the current active edge *e* is moved at the end of the *AEL* list and the whole algorithm will return back to step 2, see section 3.

If the *NAEL* list is not empty then the situation has to be solved. The point $\mathbf{p}_{min}$ with minimal distance from the centre of the bounding sphere is chosen from the *NAEL* list, see Figure 8, step 3.

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 9 illustrates the wrong situation.

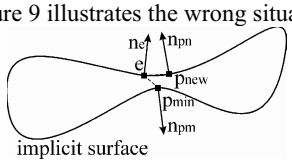

Figure 9. A problem of thin implicit objects.

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 8, 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 8, 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 8, 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 at the end of *AEL* list.

6.3. 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 7, step 2a. If the new edge does not comply a condition of surface curvature the new triangle should be split. That means, see Figure 10; if the angle α between surface normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$ at points $\mathbf{p}_{e1}$, $\mathbf{p}_{er2}$ is greater then some limit α_{split_lim} then the new triangle will be split into two new triangles, see Figure 10, step 2.

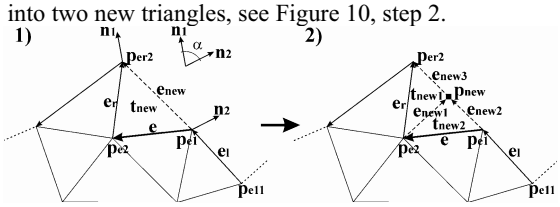

Figure 10. Splitting of the new triangle.

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 5.4.

7. 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 [12]) and the Marching cubes method (MC, Bloomenthal's polygonizer, introduced in [4]).

As a testing function, we have chosen the implicit object Genus 3 that is defined as follows.

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

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

The measured values from our experiment are in Table 1. The values have been achieved with the desired lowest level of detail (LOD) equal 0.8. It means that maximal length of triangles' edges is 0.8. Note that there is not defined a unit of length, so that number could be for example in centimeters as well as the parameters of the function Genus 3 described above.

Table 1. Values of the object Genus 3 with the desired lowest level of detail LOD = 0.8.

	AES	MTR	MC
# Triangles	4886	947	1056
# Vertices	2439	473	516
Avg dev.	10,99	56,80	73,28
Angle crit.	0,65	0,67	0,38
Elength crit.	0,77	0,78	0,54

The table contains the number of triangles and vertices generated. The value *Avg dev.* means the average deviation of each triangle from the real implicit surface. It is measured as algebraic distance of a gravity centre of a triangle from an implicit surface, i.e. the function value at the centre of gravity of the triangle. Note that the algebraic distance strongly depends on the given implicit function; in our test, the Genus 3 object is used for all methods, so the value has its usefulness. The value *Angle crit*. means the

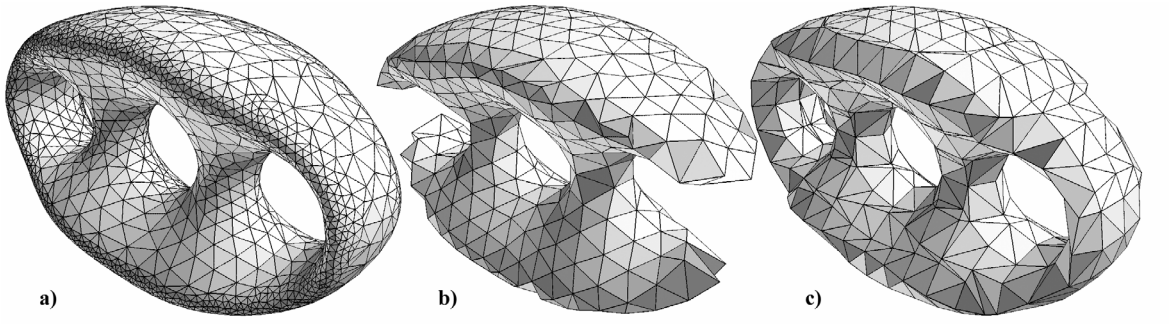

Figure 11. The Genus 3 object generated by the a) Adaptive Edge Spinning algorithm; b) Marching triangles algorithm; and c) Marching cubes algorithm.

criterion of the ratio of the smallest angle to the largest angle in a triangle and the value *Elength crit*. means the criterion of the ratio of the shortest edge to the longest edge of a triangle. The value *Avg dev.* shows the accuracy of an implicit object approximation and therefore, the Adaptive Edge spinning algorithm is logically the best of tested methods. The criterions of angles and length of edges in triangles are similar for the AES and the MTR algorithms, so the both approaches generate well-shaped triangular meshes.

For visual comparison, the resulting pictures of the Genus 3 object generated in the test are in Figure 11. Figure 11a shows the object generated by the adaptive algorithm, so the number of triangles generated is higher in dependence on the surface curvature. In Figure 11b some parts of the object are lost because the algorithm just connects nearest parts by large triangles depending of the lowest level of detail. The resulting image generated by the Marching cubes algorithm is shown in Figure 11c This algorithm produces badly-shaped triangles but it is fast and also stable for complex implicit surfaces with C^0 continuity, only.

Table 2 contains values generated by all three methods with the variable desired level of detail (LOD) because we want the number of generated triangles to be similar.

Table 2. Values of the object Genus 3 with the variable lowest level of detail.

	AES	MTR acc.	MC
LOD	0,26	0,21	0,23
# Triangles	13802	13695	13552
# Vertices	6897	6843	6756
Avg dev.	3,79	3,89	5,25
Angle crit.	0,69	0,73	0,36
Elength crit.	0,81	0,83	0,52
Time [ms]	861	101	150
TimeA [ms]	62,38	7,37	11,07

In this case, the value *Avg dev*. is similar for the AES and the MTR algorithms, because next shrinking of triangles is not necessary for achieving a better approximation. All other measured values are similar like in Table 1. The value *Time* in Table 2 shows the measured computational time of each algorithm and the value *TimeA* represents an average time needed for creating of one thousand triangles.

Note that the time values are included only for a better illustration about the algorithms because the presented method is primarily aimed at quality of approximation, not at speed. Speed-up and next improving of the algorithm will be our future work. Comparison of speed is more suitable to perform on the original non-adaptive Edge spinning method and it has been introduced in [8]. In our test, the accelerated version of the MTR method, [9], has been used and therefore, its results are better. All tests were measured on a machine AMD Athlon XP 1500+, 1GB DDR.

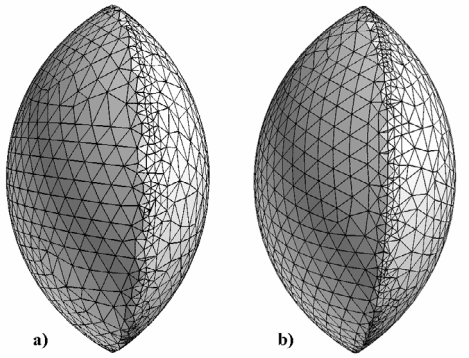

Figure 13. Intersection of two spheres generated by the Adaptive Edge spinning algorithm; with and without edge detection.

Figure 13 shows an object modeled as intersection of two spheres. The implicit object complies the C^0 continuity only and it is correctly polygonized by the proposed 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

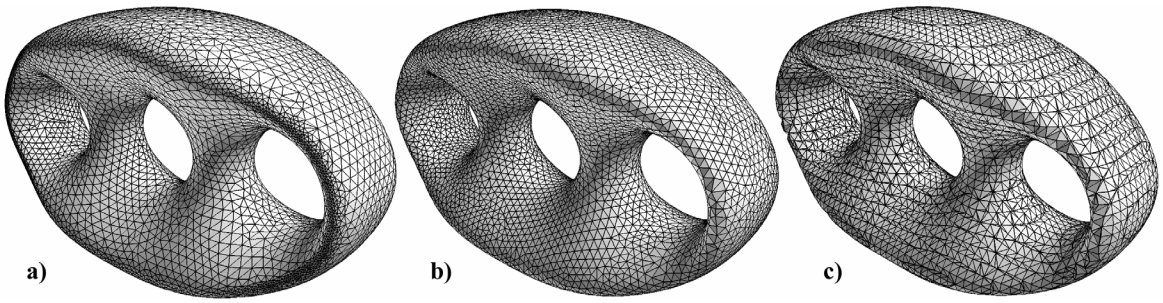

Figure 12. The Genus 3 object generated by the a) Adaptive Edge Spinning algorithm; b) Marching triangles algorithm; and c) Marching cubes algorithm.

with limit edge angle equal to $\pi/4$, see section 5.3 for details. For visual comparison, the resulting pictures of the Genus 3 object generated according to values in Table 2 are in Figure 12.

8. Conclusion

This paper presents the new adaptive approach for polygonization of implicit surfaces. The algorithm marches over the object's surface and computes the accurate coordinates of new points by spinning the edges of already generated triangles. Coordinates of the new points depend on surface curvature estimation. We used the estimation by deviation of angles of adjacent points because it is simple and fast for computation. The similar measurement has been used as curvature estimation in [23] as well. Our experiments also proved its functionality. Other estimation techniques can be found in [3], [15].

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). In future work, we want to modify the current algorithm for more complex implicit functions of the C^0 continuity, only.

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. Presented project has been implemented as a part of the MVE (Modular Visualization Environment), [16].

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

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

[6] Bloomenthal, J., Bajaj, Ch., Blinn, J., Cani-Gascuel, M-P., Rockwood, A., Wyvill, B., Wyvill, G.: *Introduction to implicit surfaces*, Morgan Kaufmann, 1997.

[7] Čermák, M., Skala, V.: *Polygonization by the Edge Spinning*, Int. Conf. Algoritmy 2002, Slovakia, 2002.

[8] Čermák, M., Skala, V.: *Accelerated Edge Spinning algorithm for Implicit Surfaces*, Int. Conf. ICCVG 2002, Zakopane, Poland, 2002.

[9] Čermák, M., Skala, V.: *Space Subdivision for Fast Polygonization of Implicit Surfaces*, Int. Conf. ECI 2002, Slovakia, 2002.

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

[11] 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.

[12] Hartmann, E.: *A Marching Method for the Triangulation of Surfaces*, The Visual Computer (14), pp.95-108, 1998.

[13] Hilton, A., Stoddart, A.J., Illingworth, J., Windeatt, T.: *Marching Triangles: Range Image Fusion for Complex Object Modelling*, Int. Conf. on Image Processing, 1996.

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

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

[16] MVE – Modular Visualization Environment project, http://herakles.zcu.cz/research.php, 2001.

[17] Ohtake, Y., Belyaev, A., Pasko, A.: *Dynamic Mesh Optimization for Polygonized Implicit Surfaces with Sharp Features*, The Visual Computer, 2002.

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

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

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

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

[22] Triquet, F., Meseure, F., Chaillou, Ch.: *Fast Polygonization of Implicit Surfaces*, Int. Conf. WSCG 2001.

[23] Velho,L.: *Simple and Efficient Polygonization of Implicit Surfaces*, Journal of Graphics Tools, 1(2):5-25, 1996.