ICCS 2004, pp. 325-334, Lecture Notes in Computer Science LNCS 3045, Springer Verlag, 2004.

Curvature Dependent Polygonization by the Edge Spinning

Martin Čermák[*] and Václav Skala

University of West Bohemia, Pilsen
Department of Computer Science and Engineering
Czech Republic
{cermakm,skala}@kiv.zcu.cz

Abstract. An adaptive method for polygonization of implicit surfaces is presented. The method insists on the shape of triangles and the accuracy of resulting approximation as well. 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 methods. The main advantages of the triangulation presented are simplicity and the stable features that can be used for next expanding.

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(**x**) = 0. There are two different definitions of implicit surfaces. The first one [2], [3] defines an implicit object as f(**x**) < 0 and the second one, F-rep [9], [11], [12], defines it as f(x) ≥ 0.

Existing polygonization techniques may be classified into three categories. Spatial sampling techniques that regularly or adaptively sample the space to find the cells that straddle the implicit surface [2], [4]. Surface tracking approaches iteratively create a triangulation from a seed element by marching along the surface [1], [2], [5], [7], [10], [16]. Surface fitting techniques [11] progressively adapt and deform an initial mesh to converge to the implicit surface.

2 Algorithm Overview

Our algorithm is based on the surface tracking scheme (also known as the continuation scheme) and therefore, there are several limitations. A starting point must be determined and only one separated implicit surface is polygonized for such

[*] This work was supported by the Ministry of Education of the Czech Republic – project MSM 235200002.

ICCS 2004, pp. 325-334, Lecture Notes in Computer Science LNCS 3045, Springer Verlag, 2004.

point. Several disjoint surfaces can be polygonized from a starting point for each of them.

The algorithm uses only the standard data structures used in computer graphics. The main data structure is an 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). 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., see [5] for details.
    b. Include the edges (e_0,e_1,e_2,) of the first triangle T_0 into the active edges list.
2. Polygonize the first active edge *e* from the active edges list.
3. Update the *AEL*; delete the currently polygonized active edge *e* and include the new generated active edge/s at the end of the list.
4. If the active edges list is not empty return to step 2.

3 Edge Spinning

The main goal of this work is a numerical stability of a surface point coordinates' computation for objects defined by implicit functions. In general, a surface vertex position is searched in direction of a gradient vector $\nabla\mathbf{f}$ of an implicit function *f*, as in [7]. In many cases, the computation of gradient of the function *f* is influenced by a major error that depends on modeling techniques used [9], [10], [11], [12], [14], [15]. 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 on a circle; therefore, 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 Fig. 2; this guarantees that the new generated triangle is well shaped (isosceles).

3.1 Circle Radius Estimation

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 Fig. 1. The normal vector $\mathbf{n}_1$ is computed at point $\mathbf{s}$ 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}.

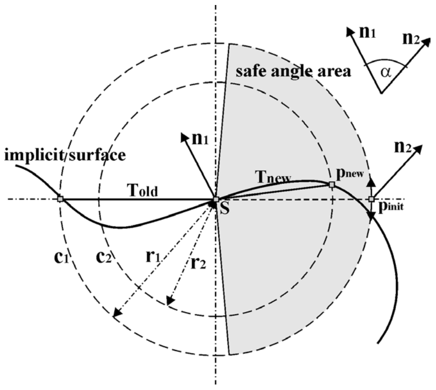

Fig. 1. The circle radius estimation.

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.

$$r_2 = r_1 \cdot k, k \in \langle 0,1 \rangle; \tag{1}$$
$$k = \left(\frac{\alpha_{\lim} - \alpha \cdot c}{\alpha_{\lim}} \right),$$

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.

3.2 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 Fig. 2. Let the angle of the initial position be $\alpha=0$.
2. Compute the function values $f(\mathbf{p}_{new}) = f(\alpha)$,
   $f(\mathbf{p}'_{new}) = f(\alpha + \Delta\alpha)$ – initial position rotated by the angle $+\Delta\alpha$,
   $f(p''_{new}) = f(\alpha - \Delta\alpha)$ - 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 $-\Delta\alpha$.
4. Let the function values $f_1 = f(\alpha)$ and $f_2 = f(\alpha \pm \Delta\alpha)$; update the angle $\alpha = \alpha \pm \Delta\alpha$.

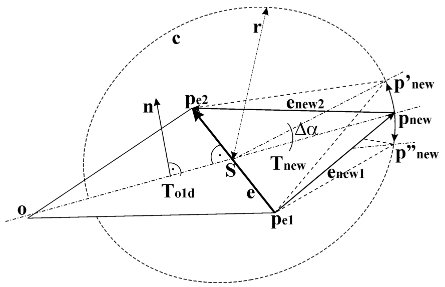

Fig. 2. The principle of root finding algorithm.

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 Fig. 1) 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.*

3.3 Root Finding of a Sharp Edge

Let us assume that the standard edge spinning root finding algorithm presented above has found the point $\mathbf{p}_{new}$. The algorithm then determines the surface normal vector $\mathbf{n}_{new}$ at this point and computes the angle α between normal vectors $\mathbf{n}_{new}$ and $\mathbf{n}_s$. The vector $\mathbf{n}_s$ is measured at mid-point **s** of the active edge *e*, see Fig. 3. 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 Fig. 3.
2. Apply the straight root finding algorithm described in section 3.4 and find the new point $\mathbf{p'}_{new}$.

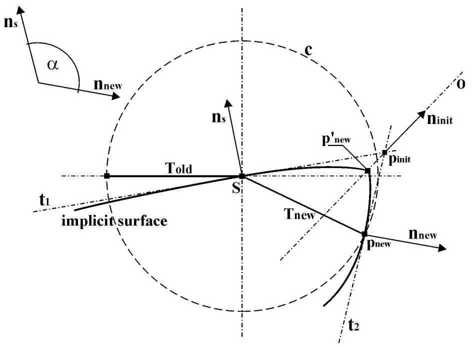

Fig. 3. The principle of root finding algorithm for sharp edges.

ICCS 2004, pp. 325-334, Lecture Notes in Computer Science LNCS 3045, Springer Verlag, 2004.

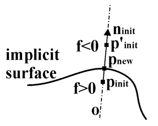

Fig. 4. Principle of root-finding in straight direction.

3.4 Straight Root Finding Algorithm

The algorithm starts from an initial point $\mathbf{p}_{init}$ (see Fig. 4) and supposes that the implicit surface is at least C^0 continuity.

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 $\mathbf{n}_{init}$ * 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.

4 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 3.2) 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 Fig. 5.

4.1 Neighborhood Test

Decision between cases a) and b) depends on relation among angles α_1, α_2, α_n, see Fig. 5, step 1; let the angle α be $min(\alpha_1,\alpha_2)$. If ($\alpha < \alpha_{shape}$) then case a) else case b), see Fig. 5, step 2; 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.

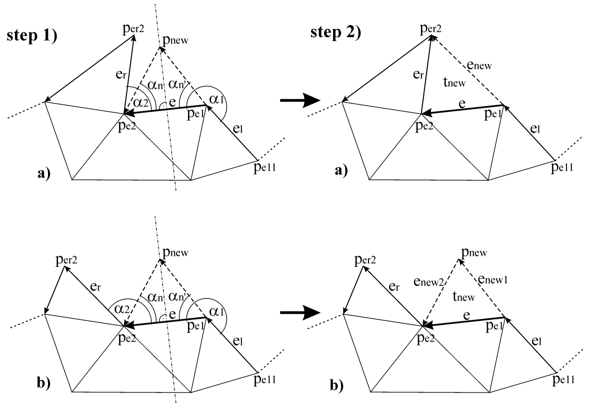

Fig. 5. Polygonization of the active edge *e*.

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.

4.2 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*). For faster evaluation of detection of global overlap there is used the space subdivision acceleration technique introduced in [6].

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 Fig. 6, 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), Fig. 5 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 4.3.
- In case b) Fig. 5 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 Fig. 5 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 2.

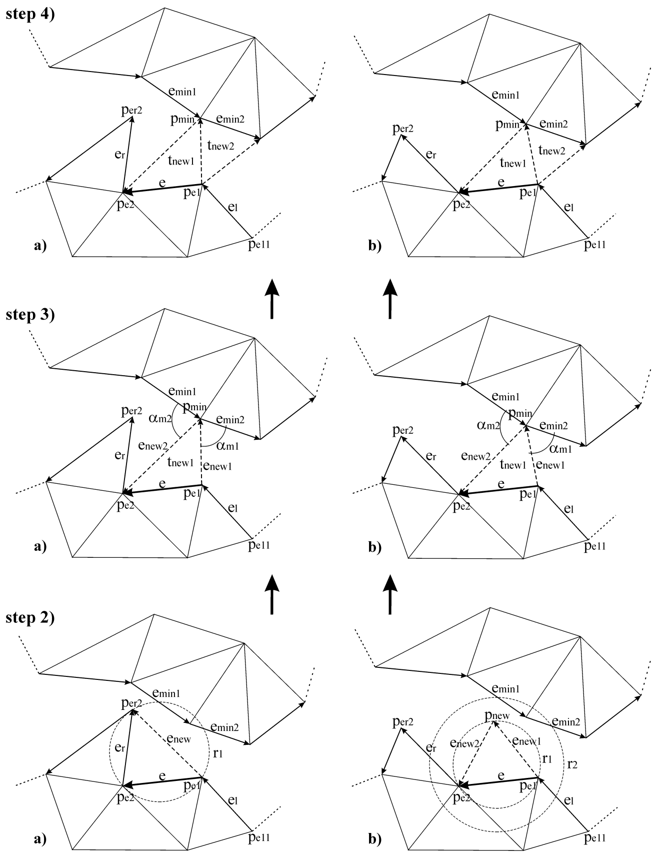

Fig. 6. Solving of distance test.

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 Fig. 6, step 3. 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 Fig. 6, 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 Fig. 6, 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, Fig. 6, step 4.

ICCS 2004, pp. 325-334, Lecture Notes in Computer Science LNCS 3045, Springer Verlag, 2004.

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.

4.3 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 Fig. 7, step 2a. If the new edge does not comply a condition of surface curvature the new triangle should be split. That means, see Fig. 7; 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 Fig. 7, step 2.

The point $\mathbf{p}_{new}$ is a midpoint of edge e_{new} and it does not lie on the implicit surface. Its correct coordinates are additionally computed by the straight root finding algorithm described in section 3.4.

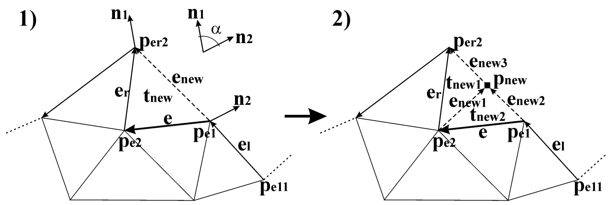

Fig. 7. Splitting of the new triangle.

5 Experimental Results

The Edge spinning algorithm (ES) 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 [7]) and the Marching cubes method (MC, Bloomenthal's polygonizer, introduced in [2]). As a testing function, we have chosen the implicit object Genus 3 that is defined as follows.

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

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

The values in Table 1 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.

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 concrete implicit function; in our test, the Genus 3 object is used for all methods, so the value has its usefulness.

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

	ES	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 value *Angle crit.* means the 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 the adaptive ES algorithm is logically the best of tested methods. The criterions of angles and length of edges in triangles are similar for the ES 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 figures below. Fig. 8a shows the object generated by the adaptive algorithm, so the number of triangles generated is higher in dependence on the surface curvature. In Fig. 8b, 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 Fig. 8c. This algorithm produces badly-shaped triangles but it is fast and also stable for complex implicit surfaces with C^0 continuity, only.

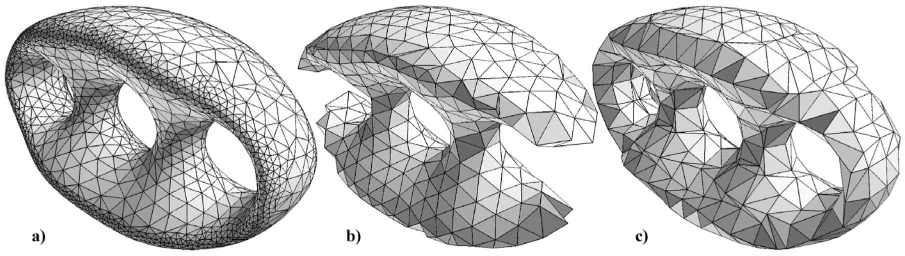

Fig. 8. The Genus 3 object generated by the a) Adaptive Edge spinning algorithm; b) Marching triangles algorithm; c) Marching cubes algorithm.

6 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 [17] as well. Our experiments also proved its functionality.

ICCS 2004, pp. 325-334, Lecture Notes in Computer Science LNCS 3045, Springer Verlag, 2004.

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.

Acknowledgement. The authors would like to thank to all who contributed to the development of this new approach, for their comments and suggestions, especially to colleagues MSc. and PhD. students at the University of West Bohemia in Plzen.

References

1. Akkouche, S., Galin, E.: Adaptive Implicit Surface Polygonization using Marching Triangles, Computer Graphic Forum, 20(2): 67–80, 2001.
2. Bloomenthal, J.: Graphics Gems IV, Academic Press, 1994.
3. Bloomenthal, J.: Skeletal Design of Natural Forms, Ph.D. Thesis, 1995.
4. Bloomenthal, J., Bajaj, Ch., Blinn, J., Cani-Gascuel, M-P., Rockwood, A., Wyvill, B., Wyvill, G.: Introduction to implicit surfaces, Morgan Kaufmann, 1997.
5. Čermák, M., Skala, V.: Polygonization by the Edge Spinning, Int. Conf. Algoritmy 2002, Slovakia, ISBN 80-227-1750-9, September 8–13.
6. Čermák, M., Skala, V.: Accelerated Edge Spinning algorithm for Implicit Surfaces, Int. Conf. ICCVG 2002, Zakopane, Poland, ISBN 839176830-9, September 25–29.
7. Hartmann, E.: A Marching Method for the Triangulation of Surfaces, The Visual Computer (14), pp. 95–108, 1998.
8. Hilton, A., Stoddart, A.J., Illingworth, J., Windeatt, T.: Marching Triangles: Range Image Fusion for Complex Object Modelling, Int. Conf. on Image Processing, 1996.
9. "Hyperfun: Language for F-Rep Geometric Modeling", http://cis.k.hosei.ac.jp/~F-rep/
10. Karkanis, T., Stewart, A.J.: Curvature-Dependent Triangulation of Implicit Surfaces, IEEE Computer Graphics and Applications, Volume 21, Issue 2, March 2001.
11. Ohtake, Y., Belyaev, A., Pasko, A.: Dynamic Mesh Optimization for Polygonized Implicit Surfaces with Sharp Features, The Visual Computer, 2002.
12. Pasko, A., Adzhiev, V., Karakov, M., Savchenko,V.: Hybrid system architecture for volume modeling, Computer & Graphics 24 (67–68), 2000.
13. Rvachov, A.M.: Definition of R-functions, http://www.mit.edu/~maratr/rvachev/p1.htm
14. Shapiro, V., Tsukanov, I.: Implicit Functions with Guaranteed Differential Properties, Solid Modeling, Ann Arbor, Michigan, 1999.
15. Taubin, G.: Distance Approximations for Rasterizing Implicit Curves, ACM Transactions on Graphics, January 1994.
16. Triquet, F., Meseure, F., Chaillou, Ch.: Fast Polygonization of Implicit Surfaces, WSCG'2001 Int.Conf., pp. 162, University of West Bohemia in Pilsen, 2001.
17. Velho,L.: Simple and Efficient Polygonization of Implicit Surfaces, Journal of Graphics Tools, 1(2):5–25, 1996.