Int.J.Computational Science and Engineering, ISSN 1742-7185, Vol.3, No.1, pp.45-52, 2007

Polygonisation of disjoint implicit surfaces by the adaptive edge spinning algorithm

Martin Čermák and Vaclav Skala*

Department of Computer Science and Engineering,
University of West Bohemia in Pilsen
Univerzitni 22, 306 14 Pilsen, Czech Republic
E-mail: cermakm@kiv.zcu.cz E-mail: skala@kiv.zcu.cz
*Corresponding author

Abstract: This paper presents an adaptive method for polygonisation of implicit surfaces. The method is based on the shape of triangles and the accuracy of resulting approximation. The main advantages of the triangulation presented are simplicity and the stable features that can be used for the next expansion. The presented algorithm is based on the surface tracking scheme, and it is compared with other algorithms that are based on the similar principle, such as the Marching cubes and the Marching triangles algorithms. Moreover, the technique for detection of more disjoint implicit surfaces in a defined area is also presented. The algorithm is not limited to a given polygonisation method, but then its use is possible for all other surface approaches that need a starting point at the beginning.

Keywords: adaptive triangulation; curvature; implicit function; polygonisation.

Reference to this paper should be made as follows: Čermák, M. and Skala, V. (2007) 'Polygonisation of disjoint implicit surfaces by the adaptive edge spinning algorithm', *Int. J. Computational Science and Engineering*, Vol. 3, No. 1, pp.45–52.

Biographical notes: Martin Čermák is a member of Computer Graphics Group at the University of West Bohemia in Pilsen. He received his PhD at the Department of Computer Science and Engineering.

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

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 visualisation.

An implicit surface is mathematically defined as a set of points in space $\mathbf{x}$ that satisfy the equation $f(\mathbf{x}) = 0$. Thus, visualising implicit surfaces typically consists in finding the zero set of f, which may be performed either by polygonising the surface or by direct ray tracing. There are two different definitions of implicit surfaces. The first one (Bloomenthal, 1994, 1995) defines an implicit object as $f(\mathbf{x}) < 0$ and the second one, F-rep (Hyperfun: Language for F-Rep Geometric Modeling, http://cis.k.hosei.ac.jp/~F-rep/; Pasko et al., 2000; Ohtake, Belyaev and Pasko, 2002), defines it as $f(\mathbf{x}) \geq 0$. In our implementation, we use the F-rep definition of implicit objects.

Existing polygonisation 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 (Bloomenthal, 1994; Bloomenthal et al., 1997). Surface tracking approaches (also known as continuation methods) iteratively create a triangulation from a seed element by Marching along the surface (Bloomenthal, 1994; Hartmann, 1998; Akkouche and Galin, 2001; Karkanis and Stewart, 2001; Triquet, Meseure and Chaillou, 2001; Čermák and Skala, 2002a-c; Araujo and Jorge, 2004). Surface fitting techniques progressively adapt and deform an initial mesh which conver the implicit surface, (Ohtake, Belyaev and Pasko, 2002). The algorithm described in Overveld and Wyvill (2004) uses a sphere object at the beginning and next applies a series of deformations to its triangulation to transform it into the required surface.

2 Data structures

The presented 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 polygonisation. 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).

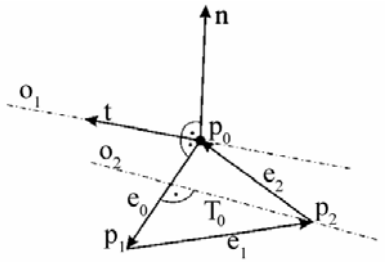
3 Principle of the algorithm

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 is polygonised for such point. Several disjoint surfaces can be polygonised from a starting point for each of them.

The whole algorithm consists of the following steps:

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

Figure 1 The first triangle



4 Initialising the polygonisation

A first triangle has to be created at the beginning of the polygonisation. The first triangle is assumed to lie near a tangent plane of the starting point $\mathbf{p}_0$. 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 **t** as in Hartmann (1998). If $(n_x > 0.5)$ or $(n_y > 0.5)$ then $\mathbf{t} = (n_y, -n_x, 0)$; else $\mathbf{t} = (-n_z, 0, n_x)$.
3 Use the tangent vector **t** as a fictive active edge and use the Edge Spinning (ES) 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.
4 Polygonise the first edge e_0 by ES 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

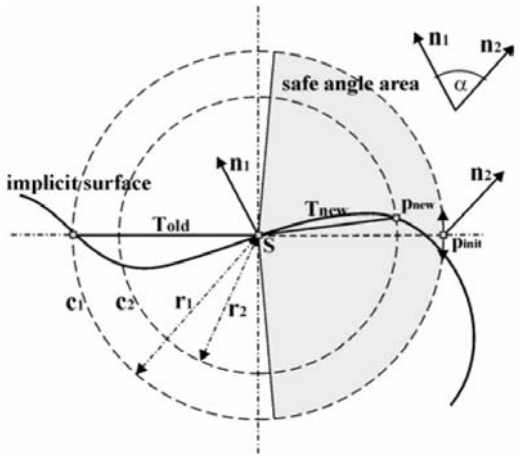
The main goal of this work is a numerical stability of a surface point coordinates' computation for objects defined by implicit functions. In general, a surface vertex position is searched in direction of a gradient vector $\nabla\mathbf{f}$ of an implicit function *f*, as in Hartmann (1998). In many cases, the computation of gradient of the function *f* is influenced by a major error that depends upon modelling techniques used (Hyperfun: Language for F-Rep Geometric Modeling, http://cis.k.hosei.ac.jp/~F-rep/; Taubin, 1994; Shapiro and Tsukanov, 1999; Pasko et al., 2000; Karkanis and Stewart, 2001; Ohtake, Belyaev and Pasko, 2002). Because of these reasons, in our approach, we have defined these restrictions for finding a new surface point $\mathbf{p}_{new}$:

1 The new point $\mathbf{p}_{new}$ is sought on a circle; therefore, each new generated triangle preserves the desired accuracy of polygonisation. The circle radius is proportional to the estimated surface curvature.
2 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 $\mathbf{n}_1$ and $\mathbf{n}_2$, 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 the beginning of polygonisation as the Lowest Desired Level of Detail (LoD).

The new circle radius r_2 is computed as follows:

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

where $\alpha_{\lim}$ 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 $\alpha = 0$.

2 Compute the function values $f(\mathbf{p}_{new}) = f(\alpha)$, $f(\mathbf{p}'_{new}) = f(\alpha + \Delta\alpha)$ – initial position rotated by the angle $+\Delta\alpha$, $f(\mathbf{p}''_{new}) = f(\alpha - \Delta\alpha)$ – initial position rotated by the angle $-\Delta\alpha$; 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 $f1 = f(\alpha)$ and $f2 = f(\alpha \pm \Delta\alpha)$; update the angle $\alpha = \alpha \pm \Delta\alpha$.

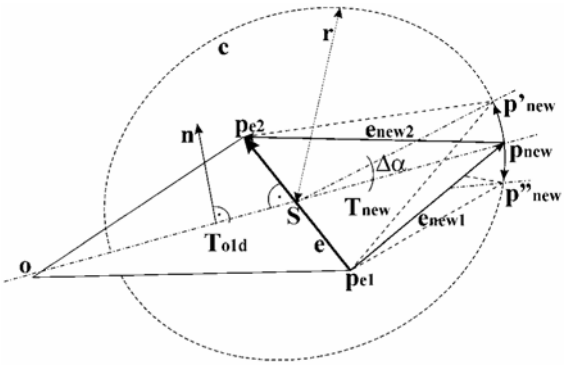
5 Check which of following case appeared:

a If $(f_1 f_2) < 0$ 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 0

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*.

Figure 3 The principle of root finding algorithm



5.3 Root finding of a sharp edge

Let us assume that the standard ES 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 than 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 $\mathbf{p}'_{new}$.

5.4 Straight root finding algorithm

The algorithm starts from an initial point $\mathbf{p}_{init}$ (see Figure 5) 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 $\mathbf{n}_{init} * \text{sign}(f(\mathbf{p}_{init}))$; 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 the 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 4 The principle of root finding algorithm for sharp edges

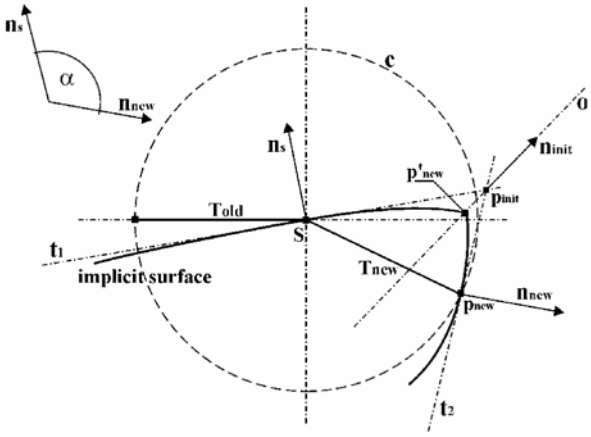
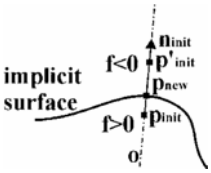


Figure 5 Principle of root finding in straight direction



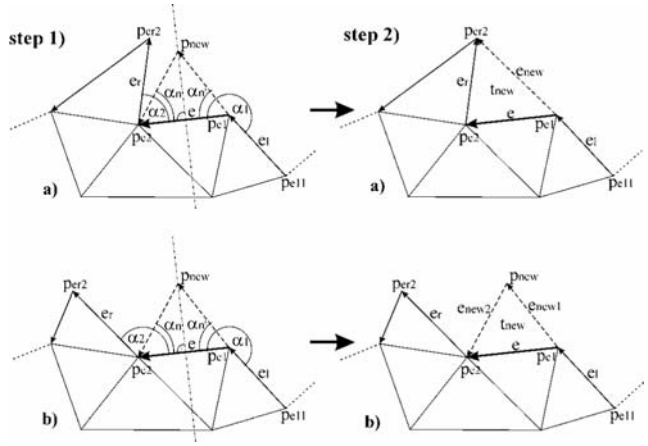
6 Polygonisation of an active edge

Polygonisation 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 6.

6.1 Neighbourhood test

Decision between cases (a) and (b) depends upon relation among angles α_1, α_2, α_n, see Figure 6, step 1; let the angle α be $min(\alpha_1, \alpha_2)$.

Figure 6 Polygonisation of the active edge e



If ($\alpha < \alpha_{shape}$) then case (a) else case (b), see Figure 6 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 than 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 neighbours, see step 2a.

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

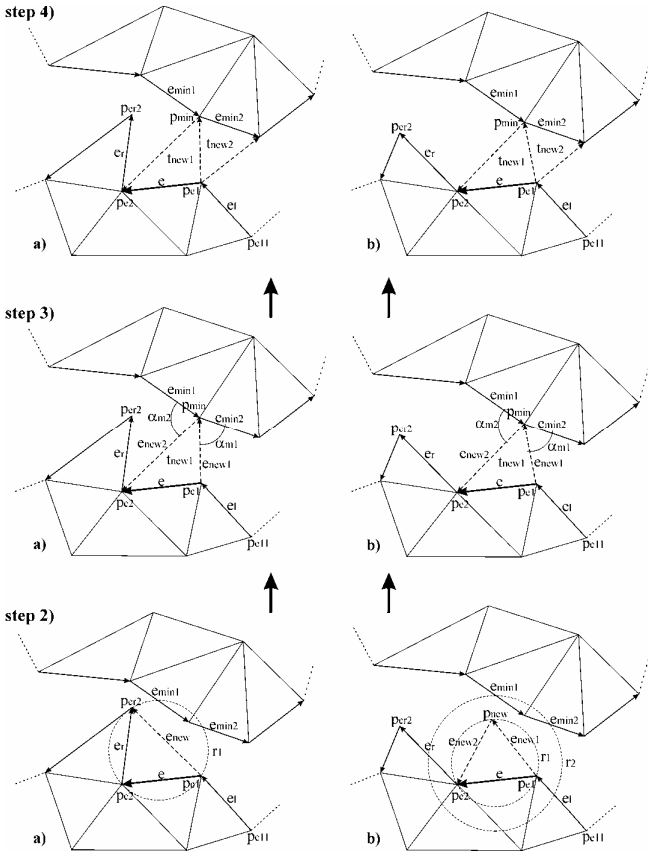
In both cases, a bounding sphere is determined for the new triangle t_{new}. The bounding sphere is the minimal sphere that contains all three points of the triangle, i.e. the centre of the sphere lies in the plane defined by these three points. If there is not a new triangle (the point $\mathbf{p}_{new}$ 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*). For faster evaluation of detection of global overlap, there is used the space subdivision acceleration technique introduced in Čermák and Skala (2002b).

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 7, 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 $c = 1.5$.

Figure 7 Solving of distance test



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

In case a, Figure 6 step 2, the current active edge e and its neighbour 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 6 step 2, the current active edge e is deleted from the list and two 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 $\mathbf{p}_{new}$ does not exist and case a in Figure 6 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 0, 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 7, 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 Figure 7, 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 neighbourhood 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 polygonised and the second one will be cached in memory for later use. The solution depends upon angles α_{m1}, α_{m2}, see Figure 7, step 3. If ($\alpha_{m1} < \alpha_{m2}$) then the left pair is polygonised; else the right pair is polygonised.

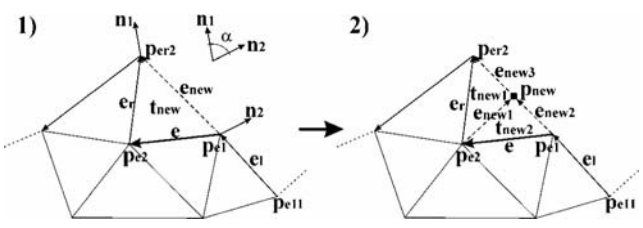
In both cases, the recently polygonised pair is automatically removed from the list and the previously cached pair of edges is returned into the list. The point $\mathbf{p}_{min}$ is contained only in one pair of active edges and the border of the triangulated area is correct, Figure 7, step 4.

Note that the polygonisation of one pair of edges consists just of joining its end points by the edge and this second new triangle has to fulfil 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 8, step 2a. If the new edge does not comply a condition of surface curvature the new triangle should be split. That means, see Figure 8; if the angle α between surface normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$ at points $\mathbf{p}_{e1}$, $\mathbf{p}_{er2}$ is greater than some limit α_{split_lim} then the new triangle will be split into two new triangles, see Figure 8, step 2.

Figure 8 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.

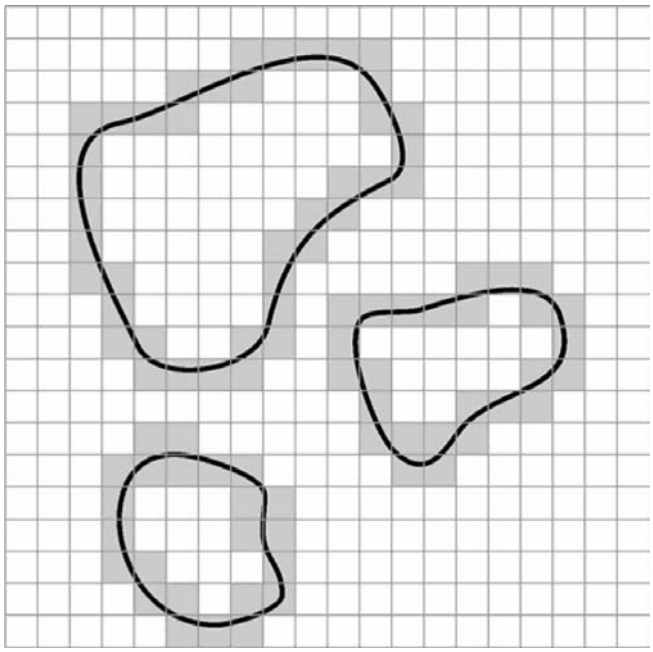
7 Detection and polygonisation of disjoint implicit surfaces in a given area

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

Because of an implicit function can be an arbitrary unknown algebraic function, there is no other way how to detect more disjoint surfaces in a defined area than use of exhaustive search approach (described for the Marching Cubes (MC) and tetrahedra methods in Bloomenthal (1994)).

Our algorithm divides the given polygonisation area by the regular grid, see Figure 9. An important note is that a size of grid cells need not to be proportional to extracted object detail but it should only be proportional to the size of the smallest object. The size of grid cells is only needed for detection of implicit components in a scene, it has no relation to object detail and this is the main difference from MC approaches. Therefore, a number of grid divisions is much lesser for our algorithm than for MC method as well as a computational time.

Figure 9 The polygonisation area divided by the regular grid that contains three implicit objects. Grid cells intersected by the implicit function are highlighted



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

1 Use ES algorithm to polygonalise the first object in the area

2 Create a function grid – find and mark all grid cells intersected by the implicit function

3 Create a triangulation grid – find and mark all grid cells intersected by the triangular mesh

4 Check if there is a marked function grid cell that has an unmarked equivalent and even unmarked neighbourhood in the triangulation grid

   a If YES – there is a new implicit component and continue as follows:

      - Find a new starting point in the given cell
      - Use the ES method for triangulation of the component
      - Return to step 3.

   b If NO – there is no other component – end of polygonisation.

7.1 Notes

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

8 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 (1998)) and MC method (Bloomenthal's polygoniser, introduced in Bloomenthal (1994)).

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

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

where 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 LoD equal to 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 the number could be for example in centimetres 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 upon 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 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 AES algorithm is logically the best of tested methods. The criterions of angles and length of edges in triangles are similar for AES and MTR algorithms, so the both approaches generate well-shaped triangular meshes.

Table 1 Values of the object Genus 3 with 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

For visual comparison, the resulting pictures of the Genus 3 object generated in the test are in figures below. Figure 10a shows the object generated by the adaptive algorithm, so the number of triangles generated is higher in dependence upon the surface curvature. In Figure 10b, some parts of the object are lost because the algorithm just connects the nearest parts by large triangles depending upon LoD. The resulting image generated by MC algorithm is shown in Figure 10c. This algorithm produces badly-shaped triangles, but it is fast and also stable for complex implicit surfaces with C^0 continuity, only.

Figure 11 shows the object modelled as intersection of two spheres. The left picture is polygonised without the edge detection, i.e. the limit edge angle $\alpha_{\text{lim\_edge}}$ is equal to π and the right picture is polygonised with the limit edge angle equal to $\pi/4$, see Section 5.3 for details. This object complies only the C^0 continuity and it is correctly polygonised by our method.

Next measured experimental results are aimed at polygonisation of unknown implicit scenes consisting of more disjoint surface components. AES algorithm will be compared with MC method – Exhaustive search (MCE).

The first scene is illustrated in Figure 12 and contains two entwined spirals.

Figure 10 The Genus 3 object generated by (a) AES algorithm; (b) MTR algorithm and (c) MC algorithm

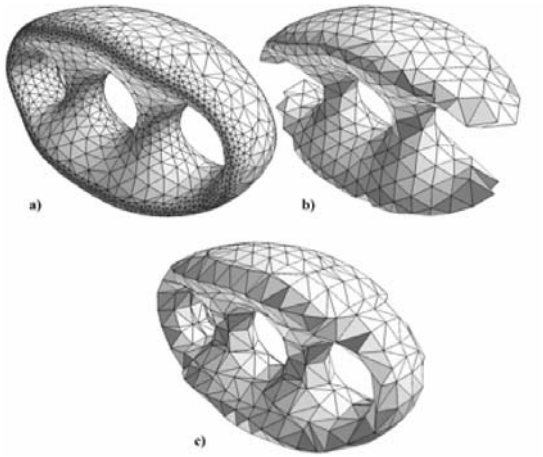


Figure 11 Intersection of two spheres generated by AES algorithm

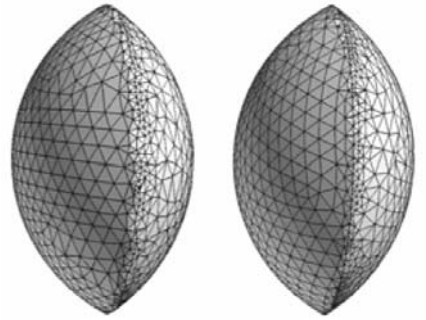


Figure 12 The Spirals model polygonised by AES spinning algorithm; (a) the first spiral and (b) both spirals polygonised

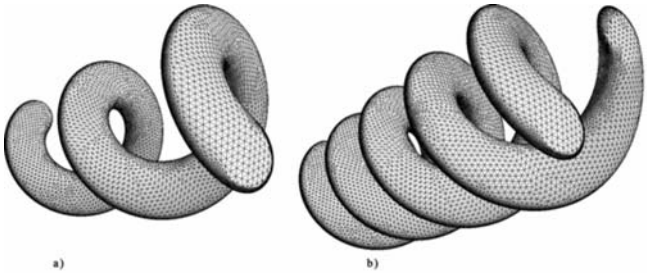
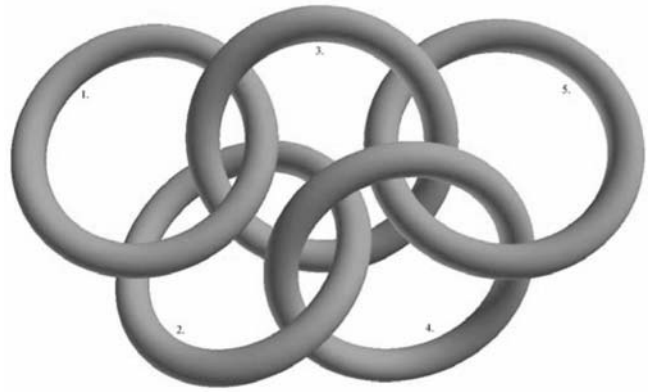


Figure 13 The Olympic rings model polygonised by AES algorithm. The numbers means the order of polygonisation



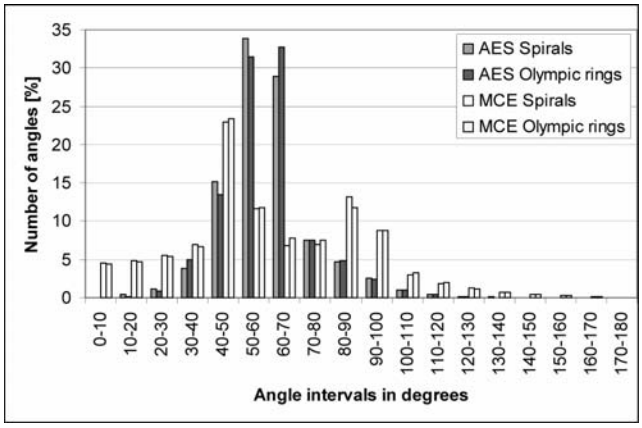
Regarding the recent Olympic Games, the second implicit scene represents the Olympic rings. The model consists of five disjoint components and has been modelled as union of five tori, see Figure 13.

The measured values from the experiment are contained in Table 2. The level of detail for both algorithms has been set so that a number of triangles as well as the approximation quality to be similar. In such case, MC method is much slower than ES algorithm and moreover, it generates poor triangular mesh, see histogram of angle distributions in Figure 14.

Table 2 Values generated by the Edge Spinning and MC algorithms

	AES		*MCE*	
	Spiral	*Olympic rings*	*Spiral*	*Olympic rings*
Subdivisions, M	50	50	400	300
LoD	0.16	0.16	0.08	0.11
Triangles	134,316	147,346	131,776	230,572
Vertices	67,163	73,673	65,892	115,286
Alg dist avg	1.79×10^{-3}	1.88×10^{-2}	2.65×10^{-3}	$1{,}51\times10^{-2}$
Angle criterion	0.70	0.69	0.36	0.37
Edge length criterion	0.81	0.80	0.52	0.53
Time [ms]	6,453	7,375	42,844	27,422
Avg time [ms]	48.04	50.05	325.13	118.93

Figure 14 Histogram of triangles shape quality



9 Conclusion

This paper presents a new adaptive approach for polygonisation 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 upon the 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

Velho (1996) as well. Our experiments also proved its functionality.

The algorithm can polygonise 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.

Moreover, a method, that makes possible polygonisation of more disjoint implicit surfaces in a defined area, has been presented as well. The algorithm is not limited to a given polygonisation technique but then, its use is possible for all other surface approaches that need a starting point at the beginning. The algorithm works well, it is able to find and polygonise all implicit components in a given region and the computational time is much less than in case of MC method – exhaustive search. All advantages of surface approaches are preserved.

Acknowledgements

The authors thank all those who contributed to the development of this new approach, especially colleagues and MSc and PhD students at the University of West Bohemia in Pilsen. This work was supported by the Ministry of Education of the Czech Republic – projects MSM 235200005 and 2C06002.

References

Akkouche, S. and Galin, E. (2001) 'Adaptive implicit surface polygonization using marching triangles', *Computer Graphic Forum*, Vol. 20, pp.67–80.

Araujo, B. and Jorge, J. (2004) 'Curvature dependent polygonization of implicit surfaces', Paper presented in the Proceedings of the *XVII Brazilian Symposium on Computer Graphics and Image Processing* (SIACG'04/SIBGRAPI'04), pp.256–273.

Bloomenthal, J. (1994) 'An implicit surface polygonizer', in P. Heckbert (Ed.), *Graphics Gems IV* (pp.324–350). Boston, MA: Academic Press.

Bloomenthal, J. (1995) 'Skeletal Design of Natural Forms', PhD Thesis, University of Calgary.

Bloomenthal, J., Bajaj, Ch., Blinn, J., Cani-Gascuel, M-P., Rockwood, A., Wyvill, B. and Wyvill, G. (1997) *Introduction to implicit surfaces*, San Francisco, CA: Morgan Kaufmann, p.332.

Čermák, M. and Skala, V. (2002a) 'Polygonization by the Edge Spinning', *16th Conference on Scientific Computing*, Algoritmy, Slovakia, ISBN 80-227-1750-9, September 8–13.

Čermák, M. and Skala, V. (2002b) 'Accelerated Edge Spinning algorithm for Implicit Surfaces', *International Conference on Computer Vision and Graphics, ICCVG*, Zakopane, Poland, ISBN 839176830-9, September 25–29.

Čermák, M. and Skala, V. (2002c) 'Space Subdivision for Fast Polygonization of Implicit Surfaces', *The Fifth International Scientific Conference, ECI*, Slovakia, ISBN 80-7099-879-2, October 10–11.

Hartmann, E. (1998) 'A marching method for the triangulation of surfaces', *The Visual Computer*, Vol. 14, pp.95–108.

'Hyperfun: Language for F-Rep Geometric Modeling', Available at: http://cis.k.hosei.ac.jp/~F-rep/.

Karkanis, T. and Stewart, A.J. (2001) 'Curvature-dependent triangulation of implicit surfaces', *IEEE Computer Graphics and Applications*, March, Vol. 21, pp.60–69.

Ohtake, Y., Belyaev, A. and Pasko, A. (2002) 'Dynamic mesh optimization for polygonized implicit surfaces with sharp features', *The Visual Computer*, Vol. 11, pp.429–446.

Overveld, K. and Wyvill, B. (2004) 'Shrinkwrap: an efficient adaptive algorithm for triangulating an iso-surface', *The Visual Computer*, Vol. 20, pp.362–379.

Pasko, A., Adzhiev, V., Karakov, M. and Savchenko, V. (2000) 'Hybrid system architecture for volume modeling', *Computer and Graphics*, Vol. 24, pp.67–68.

Shapiro, V. and Tsukanov, I. (1999) 'Implicit functions with guaranteed differential properties', *Solid Modeling*, Ann Arbor, Michigan, pp.258–269.

Taubin, G. (1994) 'Distance approximations for rasterizing implicit curves', *ACM Transactions on Graphics*, January, Vol. 13, pp.3–42.

Triquet, F., Meseure, F. and Chaillou, Ch. (2001) 'Fast Polygonization of Implicit Surfaces, WSCG'2001', *Int. Conf.*, p. 162, Pilsen, Czech Republic: University of West Bohemia.

Velho, L. (1996) 'Simple and efficient polygonization of implicit surfaces', *Journal of Graphics Tools*, Vol. 1, pp.5–25.