ECI2002, conference proceedings, Univ.of Kosice, Kosice, Slovakia, ISBN 80□7099□879□2, pp.302-307, 2002

Space Subdivision for Fast Polygonization of Implicit Surfaces

Martin Cermak[1], Vaclav Skala
Department of Computer Science and Engineering,
University of West Bohemia, Univerzitni 22, 306 14 Plzen, Czech Republic
E-mail: {cermakm|skala}@kiv.zcu.cz

Abstract

This paper presents the basic principles for the visualization of objects which are defined by implicit functions and CSG trees. The basic principles (Marching cubes, Marching tetrahedra and Marching triangles) for iso-surfaces rendering of such objects are compared. A new fast modification of the Marching triangles algorithm is presented and compared with others algorithms. It is based on the space subdivision technique that enabled a significant speed-up of the Marching triangles algorithm. The speed-up grows with the grid resolution in which the object is represented. The presented algorithm is convenient for objects with large smooth and complex surfaces. The method produces a triangular mesh that consists of well-shaped triangles.

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 $\boldsymbol{x}$ that satisfy the equation $f(\boldsymbol{x}) = 0$ [2, 3, 8, 9, 11]. 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.

2. Related work

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, 10]. 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, 7, 12].

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

3. Marching triangles

In this section, the basic principle of the original Marching triangles algorithm, described in [5], is introduced. Further, the acceleration techniques, used in our implementation, are interpreted.

3.1 Original algorithm

The idea of algorithm consists of five steps:
Step 0: Arbitrarily choose a starting point $\boldsymbol{s}$ in the neighborhood of the surface and find the point $\boldsymbol{p_1}$ that lies on the surface. Surround $\boldsymbol{p_1}$ with a regular hexagon $\boldsymbol{q_2},\ldots,\boldsymbol{q_7}$ in the tangent plane. Determine the points $\boldsymbol{p_2},\ldots,\boldsymbol{p_7}$ corresponding to the starting points $\boldsymbol{q_2},\ldots,\boldsymbol{q_7}$ that lie on the surface (Figure 1a). The triangles $(\boldsymbol{p_1}, \boldsymbol{p_i}, \boldsymbol{p_{i+1}})$ are the first six triangles of the triangulation. The ordered array of points $\boldsymbol{p_2},\ldots,\boldsymbol{p_7}$ form the first actual front polygon $\prod_0$.

Step 1: For every point of the actual front polygon $\prod_0$, determine the angle of the area till to be triangulated and form front angles (Figure 1b).

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

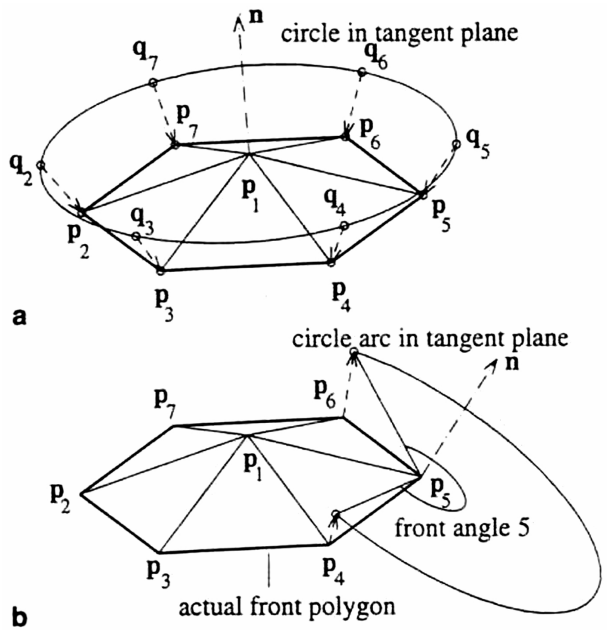

Figure 1: *The first steps of the MTR algorithm.*

Step 2: Check if any point $\boldsymbol{p_i}$ of the actual front polygon is near:

a) to a point of $\prod_0$ that is different from $\boldsymbol{p_i}$ and its neighbors. Then divide the actual front polygon $\prod_0$ into a smaller one and an additional front polygon (Figure 2a).

b) to a point of any other front polygon $\prod_m$, $m>0$. Then unite the polygons $\prod_0$, $\prod_m$ to a new and larger actual front polygon (Figure 2b). Delete $\prod_m$.

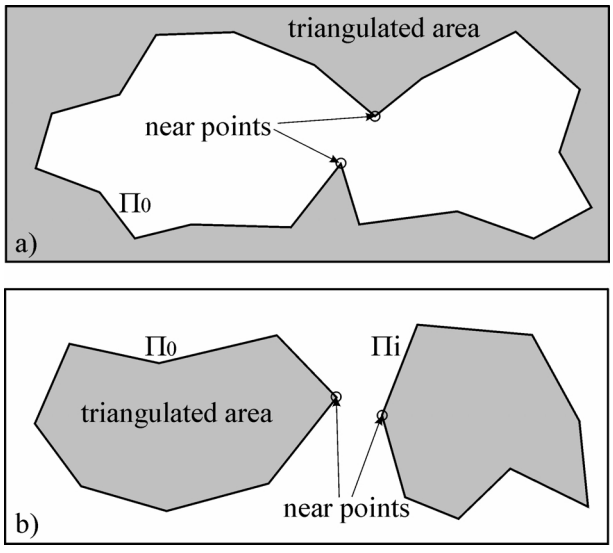

Figure 2: *(a) Dividing the actual front polygon (step 2a of the MTR algorithm) and (b) uniting two front polygons (step 2b of the MTR algorithm).*

Step 3: Determine a front point $\boldsymbol{p_i}$ of the actual front polygon $\prod_0$ with a minimal front angle. Surround $\boldsymbol{p_i}$ with triangles with angles $\approx 60^0$. Delete $\boldsymbol{p_i}$ from the actual front polygon $\prod_0$ and insert the new points into the actual front polygon $\prod_0$.

Step 4: Repeat steps 1-3 until the actual front polygon $\prod_0$ consists of only three points that generate a new triangle. If there is another (nonempty) front polygon left, it becomes the new actual front polygon $\prod_0$ and steps 1-3 are repeated. If there are no more front polygons then the triangulation is finished.

3.2 Acceleration

The MTR algorithm generates a well-shaped triangle mesh, nevertheless, the computational time of the original MTR algorithm was unacceptable and therefore the need for acceleration is obvious. There are three standard ways of speeding-up the algorithms, namely:

- decreasing the algorithm complexity and preprocessing,
- employing space subdivision techniques,
- effectively coding the linear speed-up.

The first two approaches in particular usually lead to a significant speed-up.

Decreasing the algorithm complexity

The original algorithm described in [5] contains few parts, which can be implemented more effectively. The most time-consuming part is the distances checking of the front polygons' points (Step 2 of the MTR algorithm). Our modification of the algorithm is directed precisely towards achieving this step.

The first distance check (FDC, step 2a of the algorithm) is on the complexity of the algorithm:

$$O\left(\tfrac{1}{2} m * (m-5)\right) \Rightarrow O\left(m^2\right),$$

where m is a number of points in the actual front polygon.

The second distance check (SDC, step 2b of the algorithm) is on the complexity of the algorithm:

$$O(m * u),$$

where *u* is a number of points in all the other front polygons.

Both distance checks, FDC and SDC, are evaluated at each step of the MTR algorithm and, therefore, the final algorithm complexity is:

$$O\left(\left(m^2 + m * u\right) * s\right) \Rightarrow$$
$$O\left(m * (m+u) * s\right) \approx O\left(N^3\right),$$

where *s* is a number of repetitions of the MTR algorithm.

If we realize that the shape and structure of the only actual front polygon is modified during the polygonization process and that all the others front polygons stand without any changes. Thereinafter, the actual front polygon's shape is only modified in one location where new points are included. The result of those causes is: the main part of a scene is static and only one local limited area is dynamically modified. Therefore, the distance checking need only be performed for new points and Step 2 of the MTR algorithm can then be written as:

Step 2: Check if only the **new** points $\boldsymbol{p_i}$ of the actual front polygon are near … (steps *a* and *b* are without changes).

This means that the algorithm complexity for one step and one inserted point is decreased and can be expressed for FDC as:

$$O(m),$$

where *m* is a number of points in the actual front polygon, and for SDC as:

$$O(u),$$

where *u* is a number of points in all the other front polygons.

Therefore, the final algorithm complexity is:

$$O\left((m+u) * s\right) \approx O\left(N^2\right),$$

where *s* is a number of repetitions of the MTR algorithm.

The algorithm complexity of the MTR method was reduced by one level. Therefore, the usage of the algorithm for polygonization of more detailed objects (with a large number of polygons as its output) was significantly increased. Nevertheless, our experiments proved that the MC algorithm (the surface tracking scheme [2]) is, for polygonization of highly detailed objects, significantly speedier than this modified MTR method. The reason is that the distance checking is still using large algorithm complexity for the growing number of points in front polygons.

Space subdivision scheme

One possible solution is the subdivision of the computing area into smaller sub-areas. Each sub-area contains only one part of a set of front polygons' points. The average number of points in sub-areas depends on the sub-areas' size. The main of our requirements is to minimize the number of distance checks, i.e. a selection of the most restricted set of points into which the actual front polygon can be divided or united.

The actual front polygon is divided or united only if the distance between two specified points is shorter than δ (more information in [5]). Therefore, the most suitable choice for the size of the sub-areas side is δ, i.e. the shape of sub-areas is a cube. For this choice, the distance checks (FDC and SDC) can be made only with front polygons' points which lie in the sub-areas in the neighborhood of the new point's sub-area. Figure 3 shows a distance check for a new included point (for illustration only the E^2 example).

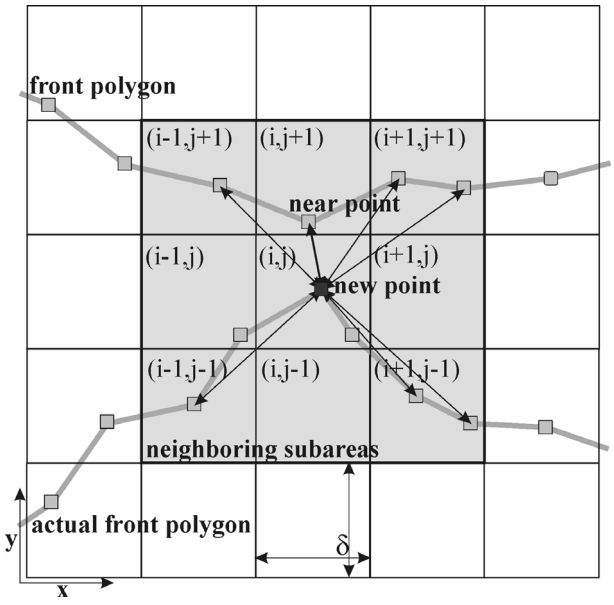

Figure 3: *The space subdivision scheme for distance checking.*

Each sub-area contains a list of front polygons' points which are located inside. The data structure, used for subdivision scheme, is shown in Figure 4.

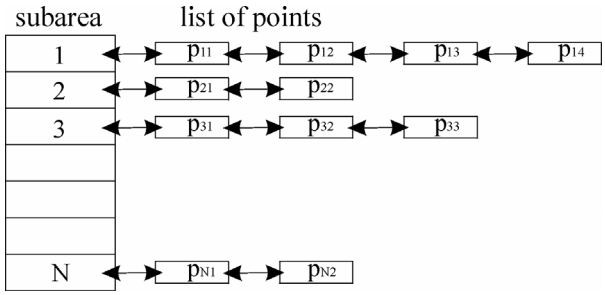

Figure 4: *The data structure for the space subdivision scheme.*

Each front polygon also has its own set of points (similar as above) and each point contains a pointer to its sub-area and a pointer to its front polygon too.

Therefore, both distance checks can be performed in one step. If both tested points lie in the actual front polygon the FDC is evaluated. In other ways, the SDC is evaluated.
It is obvious that both checks, FDC and SDC, can be performed with complexity:

$$O(v * s),$$

where *v* is the average number of points in the actual sub-area and the neighbor's sub-areas together.
The relation between *v*, *m* and *v*, *u* is:

$$(v << m) \wedge (v << u),$$

where *m* is the number of points in the actual front polygon and *u* is the number of points in all the other front polygons.

It is necessary to know that there is a trade-off between computational time and memory requirements. The memory requirement grows with the resolution of the space subdivision for the given area (space, where the surface should be extracted). In spite of this effect it is necessary to see that available memory on today's computers is growing much faster than the processors speed. It means that this kind of trade-off between runtime and memory is acceptable.

Both improvements mentioned above were implemented and verified. Experimental results are presented in next section.

4. Results

At first, the original MTR algorithm is compared to the accelerated one. The emphasis is directed towards speed comparison, because the polygonal mesh quality (the shape of triangles) is preserved for both methods, see Figure 7.
Further, the accelerated MTR algorithm is compared to the MC method (the surface tracking scheme [2]). In this experiment, we want to show that the MTR method's computational time is now close to the MC algorithm. The comparison of the output triangle mesh's quality for both the methods is illustrated in Figure 10.

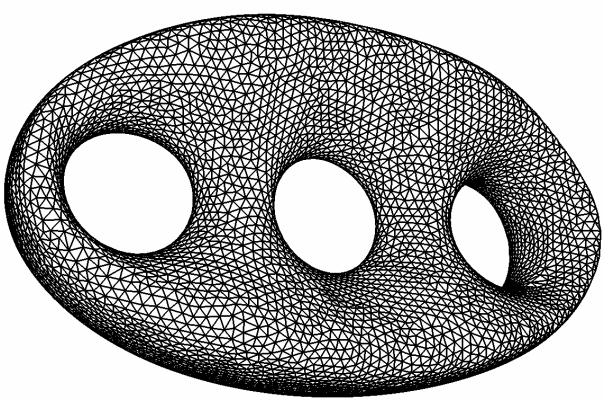

Figure 5: *The implicit object Genus, 15 679 triangles, 7 835 vertices.*

All the experiments described above were accomplished on implicit object Genus, Figure 5. The implicit function of the Genus object is written as:

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

where $r_x=6$; $r_y=3,5$; $r_z=4$; $r_1=1,2$; $x_1=3,9$.

The Genus object is also introduced in [5] as an example; therefore, we used it for illustration too.
Figure 6 shows the speed-up of the original MTR algorithm and the new MTR algorithm. It can be seen that speed-up grows with the resolution linearly in the range of resolution used for experiments. The experiments proved that the proposed algorithm is especially convenient for cases where highly detailed objects are to be generated. Nevertheless, even for small resolution the proposed algorithm is significantly faster than the original one [5].

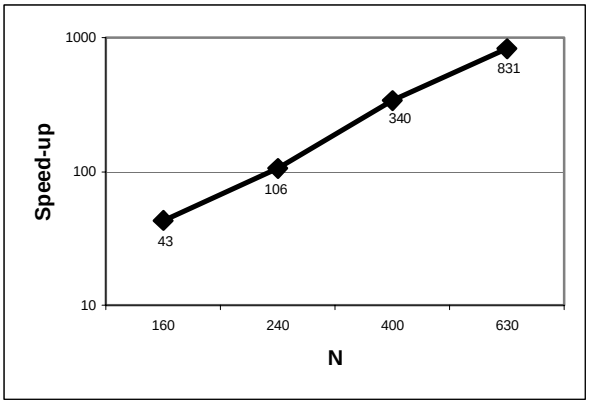

Figure 6: *Comparison of acceleration between the original MTR algorithm and the modified MTR method.*

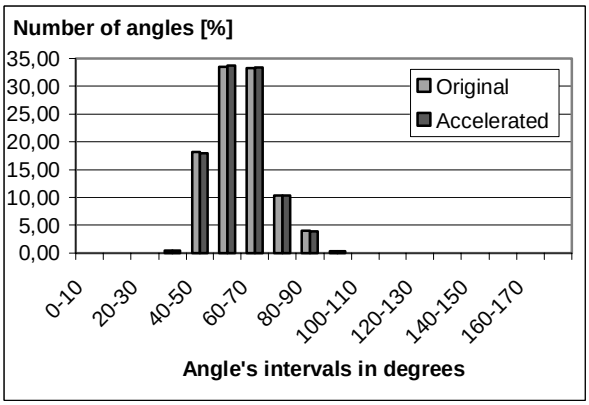

Figure 7: *Histogram of the triangle shape quality for the MTR algorithms. Generated for N=630, see Table 2.*

The size of the computing area for all experiments is shown in Table 1. The computing area is a place in E^3 where the polygonization process runs.

Axis	min	max
x	-16	16
y	-16	16
z	-16	16

Table 1: *The computing area size.*

Table 2 contains the list of values which were obtained by the first experiment. It is obvious that both modifications of the MTR algorithm generate comparable values (number of triangles and vertices) only the computational time is significantly different.

Note, that the *N* values, used in tables bellow, determine the number of sub-areas in each axis of the computing area. Therefore, the size of each sub-area (δ, see Figure 3) and also the object detail (a number of generated triangles and vertices) are proportional to the computing area's size.

	N	160	240	400	630
a)	Triangles	15535	34945	97785	244295
	Vertices	7763	17468	48886	122143
	t [ms]	5147	27600	340459	2263275
b)	Triangles	15679	35067	97867	244103
	Vertices	7835	17529	48929	122047
	t [ms]	120	260	1001	2724

Table 2: *Comparison between a) the original MTR algorithm and b) the accelerated one.*

The next our requirement was the computational time approximation of the MTR algorithm to the MC method. Figure 8 illustrates the time ratio between those algorithms. The MC algorithm is still faster than the MTR method but this drawback is balanced by the output triangle shape quality, see Figure 9 and Figure 10.

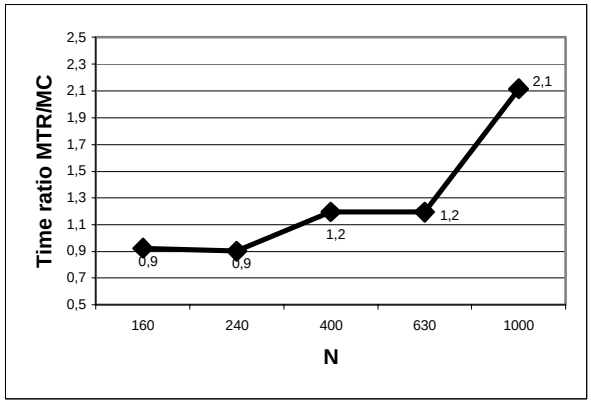

Figure 8: *Time comparison between the accelerated MTR algorithm and the MC algorithm.*

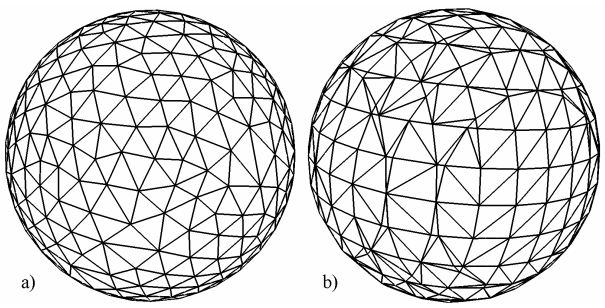

Figure 9: *Sphere polygonization: a) by the Marching triangles algorithm, b) by the Marching cubes algorithm.*

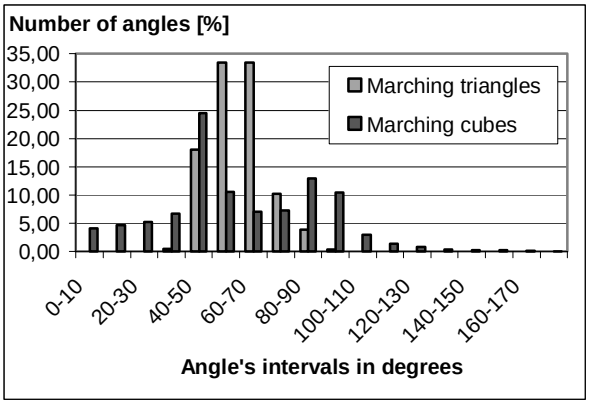

Figure 10: *Histogram of triangle shape quality between the new MTR algorithm and the MC method. Generated for N=1000, see Table 3.*

Table 3 contains the values from the second experiment, i.e. values which are used in both figures above.

	N	160	240	400	630	1000
a)	Tr.	15679	35067	97867	244103	613699
	Vert.	7835	17529	48929	122047	306845
	t [ms]	120	271	1031	2714	15402
b)	Tr.	17568	39520	109608	271344	684016
	Vert.	8772	19756	54800	135668	342004
	t [ms]	130	301	862	2273	7281

Table 3: *Comparison between a) the modified MTR algorithm and b) the MC algorithm.*

5. Conclusion

The new approach for the MTR algorithm acceleration was presented. The proposed algorithm has been tested on many non-trivial implicit surfaces and proved to have very good properties including stability for smooth[2] surfaces. Also angle distribution in the triangles generated by the MTR is better than that of the original algorithm as well. The main advantage of this new algorithm is that speed-up grows as the increases scene detail.

Acknowledgements

The authors would like to thank all those who contributed to this work especially to colleagues - MSc. and PhD. students at the University of West Bohemia in Plzen - who have stimulated the work and development of this new approach [13]. We would also like to thank Erich Hartmann for some illustrations.

[2] C^1 continuity in each point

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] Hartmann, E.: A marching method for the triangulation of surfaces, The Visual Computer (14), pp.95-108, 1998.

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

[7] Hilton, A., Illingworth, J.: Marching Triangles: Delaunay Implicit Surface Triangulation.

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

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

[10] Ohraje, Y., Belyaev, A., Pasko, A.: Dynamic meshes for accurate polygonization of implicit surfaces with sharp features, Shape Modeling International 2001, IEEE, 74-81.

[11] Uhlir, K.,Skala, V.: Interactive system for generating and modeling implicit functions, submitted for publication.

[12] Triquet, F., Meseure, F., Chaillou, Ch.: Fast Polygonization of Implicit Surfaces, WSCG'2001 Int.Conf., pp. 162, University of West Bohemia in Pilsen, 2001.

[13] Rousal, M., Skala, V.: Modular Visualization Environment - MVE, Int. Conf. ECI 2000, Herlany, Slovakia, pp.245-250, ISBN 80-88922-25-9.