

# A Non-uniform Binary Space Partition Algorithm for 2D Implicit Curves

Francisco Morgado and Abel Gomes

Networks and Multimedia Group – Institute for Telecommunications (IT)  
Department of Informatics, University of Beira Interior, 6200-001 Covilhã, Portugal  
fmorgado@di.estv.ipv.pt, agomes@di.ubi.pt

**Abstract.** Current graphical systems include primitives to draw straight-line segments, circles, Bézier curves and surfaces, NURBS (Non-Uniform Rational B-Splines), but many of them fail displaying curve singularities (self-intersections) correctly. This paper introduces a fast and robust non-uniform binary space partition (BSP) algorithm for implicit curves possibly with self-intersections and other differentiable singularities. These singularities are computed without using traditional differential techniques.

## 1 Introduction

An implicit curve  $\mathcal{C}$  is a level set (or zero set) of some analytic function  $f$  from  $\mathbb{R}^n$  to  $\mathbb{R}$ , say  $\mathcal{C} = \{\mathbf{x} \in \Omega \subseteq \mathbb{R}^n : f(\mathbf{x}) = 0\}$ . In this paper, we consider the problem of representing 2D implicit curves defined by analytic functions. In other words, we aim to compute a polygonal approximation for a curve  $\mathcal{C} = \{(x, y) \in \Omega \subseteq \mathbb{R}^2 : f(x, y) = 0\}$  defined implicitly by some analytic function  $f : \Omega \subseteq \mathbb{R}^2 \rightarrow \mathbb{R}$ . Note that the corresponding algorithm applies not only to algebraic curves as in [4], but also to more general analytic curves (e.g. curves defined by transcendental functions). Basically, there are three categories of algorithms to represent implicit curves on a display screen, namely:

- *Representation conversion.* These algorithms convert an implicit curve into a parametric curve in order to easily display it on a screen [2], [5], [11]. However, rarely there is a global parameterization for an implicit curve, unless it is regular. In fact, a local parameterization always exist in a neighborhood of a regular point of an implicit curve, i.e. a point  $\mathbf{p} = (u, v)$  such that  $f(\mathbf{p}) = 0$  and  $\nabla f \neq 0$ . In this way, we can render a regular implicit curve through rendering algorithms for parametric curves.
- *Curve tracking.* The idea behind this class of algorithms is to follow the curve point after point in a way similar to parametric curves [7], [12]. This approach has its roots in the Bresenham’s algorithm for rendering circles, which is basically a continuation method in image space. A simple continuation method consists of integrating the Hamiltonian vector field  $(-\partial f / \partial y, \partial f / \partial x)$ , combining a simple numerical integration method with a Newton corrector [1]. These methods are attractive because they concentrate effort where it is

needed, and may adapt the computed approximation to the local geometry of the curve. Unfortunately, they need a starting point on each component of the curve.

- *Space subdivision.* It consists of dividing the ambient space into subspaces, discarding those not intersecting the curve. The subdivision is done recursively and terminates when the resulting approximation to the curve by a set of small subspaces (e.g. small rectangles) [5], [6], [15] is good enough. Robust algorithms can be implemented by using algebraic techniques and interval arithmetic [14], algebraic or rational techniques [9], [8], and floating-point arithmetic [13]. However, the space subdivision techniques were not attractive for many researchers because of their low speed-up for most applications [14]. But, recently [10] presented a technique using interval arithmetic that allows speeding up the curve rendering.

This paper deals with a new space subdivision algorithm for planar implicit curves. Unlike other space subdivision algorithms, it is shape-adaptive in the sense that there are more space subdivisions where the shape of the curve changes more significantly. As a consequence, the space subdivision is not uniform. The paper is organized as follows. Section 2 describes the non-uniform binary space partition, including the BSP data structure. Section 3 describes the continuation of points embedded in the BSP data structure that enables rendering the curve. Section 4 shows some relevant experimental results. At last, some conclusions are drawn in Section 5.

## 2 Non-uniform Binary Space Partition

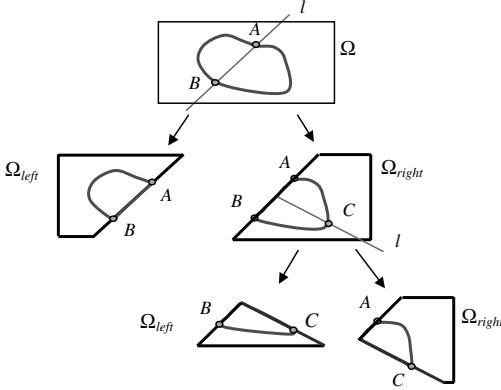
### 2.1 BSP Data Structure

Fig. 1 illustrates the non-uniform BSP (Binary Space Partition) technique for a planar implicit curve  $\mathcal{C}$ . It recursively splits up the subspace  $\Omega \in \mathbb{R}^2$  where the curve lies in into two spaces  $\Omega_{left}$ ,  $\Omega_{right}$  by some straight-line  $l$  intersecting the curve, called the BSP line. If a subspace contains a curve segment, it is partitioned again, unless the distance between the endpoints of the curve segment is less than or equal to a small tolerance  $\epsilon$ , or the curvature along the curve segment does not change too much. Subspaces without any curve segment are discarded, and those where the curvature of the curve changes are further split up. This way, our technique adapts to the shape of the curve.

In terms of C++ code, the corresponding BSP data structure is as follows:

```
class BSP {
    List *fr;
    List *lip;
    BSpline *l;
    BSP *left, *right;
    BSP *next;
}
```

The variable  $\mathbf{fr}$  denotes the frontier  $\text{Fr}(\Omega)$  of a convex subspace  $\Omega$ . It consists of a list of straight-line segments bounding  $\Omega$ ;  $\mathbf{lip}$  is a list of points resulting from the intersection  $\text{Fr}(\Omega) \cap \mathcal{C}$ ;  $\mathbf{l}$  is the line that splits up  $\Omega$  into two subsidiary subspaces  $\Omega_{left}$  and  $\Omega_{right}$ ;  $\mathbf{left}$  denotes  $\Omega_{left}$ , while  $\mathbf{right}$  denotes  $\Omega_{right}$ ;  $\mathbf{next}$  is used for rendering  $\mathcal{C}$  and represents the next left leaf of the BSP tree as explained further ahead.

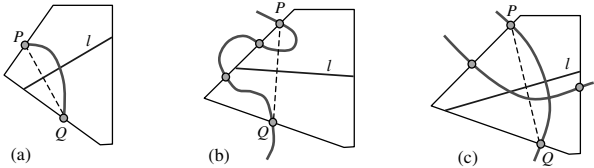


**Fig. 1.** The BSP tree.

**2.2 BSP Line**

Determining a BSP line involves the following computations on the frontier of the subspace to be split up:

- Determine the intersection points  $\text{Fr}(\Omega) \cap \mathcal{C}$ ;
- Choose the most distant points  $P, Q$  amongst those in  $\text{Fr}(\Omega) \cap \mathcal{C}$ ;
- Determine the mediatrix  $l$  of the segment  $\overline{PQ}$ .



**Fig. 2.** BSP lines for three subspaces.

The BSP line is precisely the mediatrix of  $\overline{PQ}$  (Fig. 2). The idea behind the choice of the most distant points  $P, Q$  is to partition the subspace equally as much as possible. The result is a more balanced BSP tree data structure. This eventually prevents memory stack overflow.

The corresponding algorithm to compute the BSP line is then as follows:

**Algorithm 1** (BSPLINE)

INPUT:

- (a)  $\Omega$ : a subspace of  $\mathbb{R}^2$

OUTPUT:

- (a)  $l$ : a BSP line

**Begin**

1. **if** ( $\Omega$  is the initial subspace)
  - $l \leftarrow$  an arbitrary line through the center of  $\Omega$  intersecting  $\mathcal{C}$
2. **else**
  - (a)  $lip \leftarrow \text{Fr}(\Omega) \cap \mathcal{C}$
  - (b) **if** ( $lip$ )
    - $l \leftarrow$  an arbitrary line through the center of  $\Omega$
  - (c) **else**
    - determine the most distant points  $P, Q \in lip$
    - $l \leftarrow$  mediatrix of  $\overline{PQ}$
    - **if** ( $d(P, Q) < \epsilon$ ) **return** NULL
    - **if** ( $d(P, Q) < \tau$ ) and ( $\#(\text{Fr}(\Omega) \cap \mathcal{C}) == 2$ )
      - $R \leftarrow l \cap \mathcal{C}$
      - **if** ( $\angle(\overline{RP}, \overline{RQ}) \approx 180^\circ$ ) **return** NULL
3. **return**  $l$

**End**

The BSP technique does not depend on the existence of curve singularities such as cusps and self-intersections. Besides, if a subspace contains a self-intersection, its recursive partition tends to converge to such a singularity. Therefore, it would be interesting to exploit the non-uniform BSP technique to determine self-touching points and self-intersections somehow. For that, we would start by identifying subspace leaves of the BSP tree with at least three curve points in its frontier, i.e.  $\#(\text{Fr}(\Omega) \cap \mathcal{C}) \geq 3$ . However, for intermediate nodes, this is not a valid condition for the existence of, for example, a self-intersection as illustrated in Fig. 2(b). Singularities can be only determined in subspace leaves. In fact, subspace leaves are small enough that if the condition  $\#(\text{Fr}(\Omega) \cap \mathcal{C}) \geq 3$  is true, we have a singularity in it surely. Note that subspace leaves are small because they satisfy the condition  $d(P, Q) < \epsilon$ , i.e. the distance between the most distant points  $P, Q \in \text{Fr}(\Omega) \cap \mathcal{C}$  is less than  $\epsilon$ . Thus, a partition stopping criterion is that the maximum distance between any two curve endpoints within a subspace is less than a small  $\epsilon$ , as shown in BSPLINE algorithm. Under these

conditions, we assume that a self-touching point, and also a self-intersection point, can be approximated to the midpoint of  $\overline{PQ}$ .

A second partition stopping criterion has to do with the control of the curvature. This is done by the conditions  $d(P, Q) < \tau$  and  $\#(\text{Fr}(\Omega) \cap \mathcal{C}) == 2$  in BSPLINE algorithm, where  $\tau = 3\epsilon$ . In this case, if the angle  $\angle(\overline{RP}, \overline{RQ}) \approx 180^\circ$ , the subspace partition stops, i.e. no BSP line is returned. (The point  $R$  is the intersection point between the mediatrix  $l$  and the curve  $\mathcal{C}$ .) Otherwise, the BSP line previously determined is returned and the subspace partition proceeds.

### 2.3 Position of a Curve Point in Relation to the BSP Line

Before splitting up a subspace  $\Omega$  into two subspaces  $\Omega_{left}$ ,  $\Omega_{right}$  by a straight line  $l$ , we have to classify the curve points intersecting the frontier of  $\Omega$  as belonging to either  $\Omega_{left}$  or  $\Omega_{right}$ .

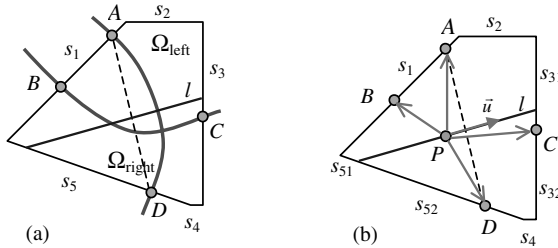


Fig. 3. Positioning curve points in relation to the BSP line.

Let  $P$  be an intersection point of the curve with the BSP line  $l$  (Fig. 3). The classification algorithm is as follows:

- Determine the unit vector  $\vec{u}$  at  $P$  with the direction of  $l$ ;
- Determine the unit vector  $\vec{w}$  orthogonal to  $\Omega$ ;
- Determine the unit vector  $\overrightarrow{PX}$  for any curve point  $X$  intersecting the frontier of  $\Omega$ ;
- If the mixed product  $(\vec{U} \times \vec{V}) \cdot \vec{w} > 0$ , with  $\vec{U} = \vec{u} \times \overrightarrow{PX}$  and  $\vec{V} = \vec{u} \times \overrightarrow{PY}$ , the points  $X, Y$  belong to the same subspace, either  $\Omega_{left}$  or  $\Omega_{right}$ ;
- If the mixed product  $(\vec{U} \times \vec{V}) \cdot \vec{w} < 0$ , with  $\vec{U} = \vec{u} \times \overrightarrow{PX}$  and  $\vec{V} = \vec{u} \times \overrightarrow{PY}$ , the points  $X, Y$  belong to distinct subspaces.

In Fig. 3, the points  $A, B$  belong to  $\Omega_{left}$ , while  $C, D$  belong to  $\Omega_{right}$ . If the mixed product  $(\vec{U} \times \vec{V}) \cdot \vec{w} = 0$ , the point  $X$  belongs to both  $\Omega_{right}$  and  $\Omega_{left}$ . This happens when  $X$  (e.g.  $P$  in Fig. 3)(b) is an intersection point between the curve and the BSP line.

### 2.4 Creation of the Subsidiary Subspaces $\Omega_{right}$ and $\Omega_{left}$

After classifying the points  $\text{Fr}(\Omega) \cap \mathcal{C}$  as belonging to either  $\Omega_{right}$  or  $\Omega_{left}$ , we have to form these subsidiary subspaces. This is equivalent to form their frontiers. This is illustrated in Fig. 3, where  $\text{Fr}(\Omega) = \{s_1, s_2, s_3, s_4, s_5\}$  is the frontier of a convex subspace  $\Omega$  in  $\mathbb{R}^2$  and  $l$  a straight-line intersecting  $\text{Fr}(\Omega)$  at exactly two points. The first intersection point subdivides  $s_3$  into two smaller segments,  $s_{31}$  and  $s_{32}$ , while the second subdivides  $s_5$  into  $s_{51}$  and  $s_{52}$ . These two intersection points originate the segment  $s_l \subset l$  that splits up  $\Omega$  into  $\Omega_{right} = \{s_l, s_{32}, s_4, s_{52}\}$  and  $\Omega_{left} = \{s_l, s_{51}, s_1, s_2, s_{31}\}$ .

Of course the creation of both subspaces requires the classification of its bounding segments in relation to the splitting line  $l$ . The segment classification is based on the point classification described in the previous subsection. The mixed product  $(\vec{U} \times \vec{V}) \cdot \vec{w}$  is greater or equal to zero for any two segment endpoints bounding the same subspace, and less than zero for endpoints belonging to distinct subspaces.

The corresponding algorithm to compute the subsidiary subspaces is:

**Algorithm 2** (SUBSPACES)

INPUT:

- (a)  $l$ : the partition line of  $\Omega$
- (b)  $\Omega$ : a subspace of  $\mathbb{R}^2$

OUTPUT:

- (a)  $\Omega_{left}$ :  $\Omega_{left} \subset \Omega$
- (b)  $\Omega_{right}$ :  $\Omega_{right} \subset \Omega$

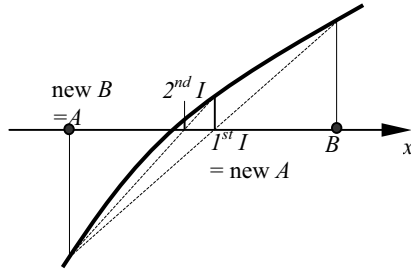
**Begin**

1.  $\Omega_{left} \leftarrow \emptyset$
2.  $\Omega_{right} \leftarrow \emptyset$
3. Determine two intersection points  $\text{Fr}(\Omega) \cap l$
4. Classify points  $\text{Fr}(\Omega) \cap \mathcal{C}$  and  $\text{Fr}(\Omega) \cap l$
5. Update frontier points  $\Omega_{left}$
6. Update frontier points  $\Omega_{right}$

**End**

### 2.5 Intersection between a Curve and a Straight-Line

The secant method is a root-finding algorithm which assumes that a function to be approximately linear in the region of interest [3]. Otherwise, there is no guarantee that it converges. Basically, it starts from two distinct estimates  $A$  and  $B$  for the root of  $f(x) = 0$ . An iterative process involving linear interpolation



**Fig. 4.** Illustration of the secant approximation method.

updates  $A$  and  $B$ , but only the most recent estimate, say  $A$ , is retained. The interpolation is given by the formula

$$I = B - f(B) \cdot \frac{(B - A)}{f(B) - f(A)} \tag{1}$$

being  $I$  is the next point that approximates the intersection point.

The secant approximation method is part of the algorithm that computes the intersection point between the curve  $\mathcal{C}$  and a subspace mediatrix  $l$ . Initially, the mediatrix is subdivided into several smaller segments. Then, the secant method is applied to each small segment that satisfies the condition  $f(A) \cdot f(B) < 0$ . This condition guarantees that there is an intersection point  $\mathcal{C} \cap l$  between  $A$  and  $B$ . The division of the mediatrix into smaller segments is needed to guarantee that the secant method converges for each segment satisfying the above condition, and also because it there may be two or more intersection points.

Let  $MAXLENGTH$  be the length of the admissible longest mediatrix, i.e. the length of the diagonal of the initial space  $\Omega$ , and  $MAX = 10$  be the maximum number of segments in such a longest mediatrix. The subdivision of a subspace mediatrix is adaptive in the sense that the number of its subsidiary segments depends on its  $LENGTH$ . If  $LENGTH \leq (MAXLENGTH/MAX)$ , then one applies the secant method to the mediatrix itself; otherwise, one subdivides the mediatrix into a number of segments given by

$$NSEG = (MAX \cdot LENGTH) / MAXLENGTH \tag{2}$$

applying then the secant method to each segment.

### 3 Continuation and Rendering of the Curve

To render an implicit curve, we have some mechanism to traverse the BSP tree in order to sequence the computed points of  $\mathcal{C}$ . Looking at Fig. 1, we see that we have only to sequence the left leaves of the BSP tree. These left leaves are collected into a list to speed up the rendering process. This has the advantage

that there is no need to visit all tree nodes. Visiting only the left leaves represents a gain of 50% in rendering  $\mathcal{C}$  in relation to a full traversal of the BSP tree. In fact, if the BSP tree goes down to the level  $N$ , its overall number of nodes is  $2^{N+1} - 1$ , but only  $2^N$  leaves are visited.

The overall rendering algorithm for implicit curves is as follows:

**Algorithm 3 (CURVE)**

INPUT:

- (a)  $\mathcal{C}$ : the curve

**Begin**

1.  $\Omega \leftarrow$  a rectangular subspace of  $\mathbb{R}^2$ ;
2.  $\text{BSP}(\Omega, 7)$ ;
3. Render  $\mathcal{C}$ ;

**End**

The sub-algorithm BSP (step 2) is the main part of the non-uniform binary space partition algorithm. It can be described as follows:

**Algorithm 4 (BSP)**

INPUT:

- (a)  $\Omega$ : a subspace of  $\mathbb{R}^2$   
 (b)  $d$ : degree of BSP tree or recursion level

**Begin**

1. **if** ( $d == 0$ ) **return** NULL;
2.  $l \leftarrow \text{BSPLINE}(\Omega)$
3. **if** ( $l$ )
  - $\text{SUBSPACES}(l, \Omega, \Omega_{left}, \Omega_{right})$ ;
  - $\text{BSP}(\Omega_{left}, d - 1)$ ;
  - $\text{BSP}(\Omega_{right}, d - 1)$ ;

**End**

Note that there is a third partition stopping criterion given by the recursion level or BSP tree degree, which is assumed to be equal to 7.

## 4 Experimental Results

Although our algorithm uses a space subdivision technique, it proved to be fast. Unexpectedly, we noted that the algorithm performs faster where the shape of the curve changes more significantly. This happens because the number of subspaces increases where the shape of the curve is non-symmetric or irregular. The curves pictured in Fig. 5 were drawn with a precision of  $\epsilon = 10^{-3}$  and maximum

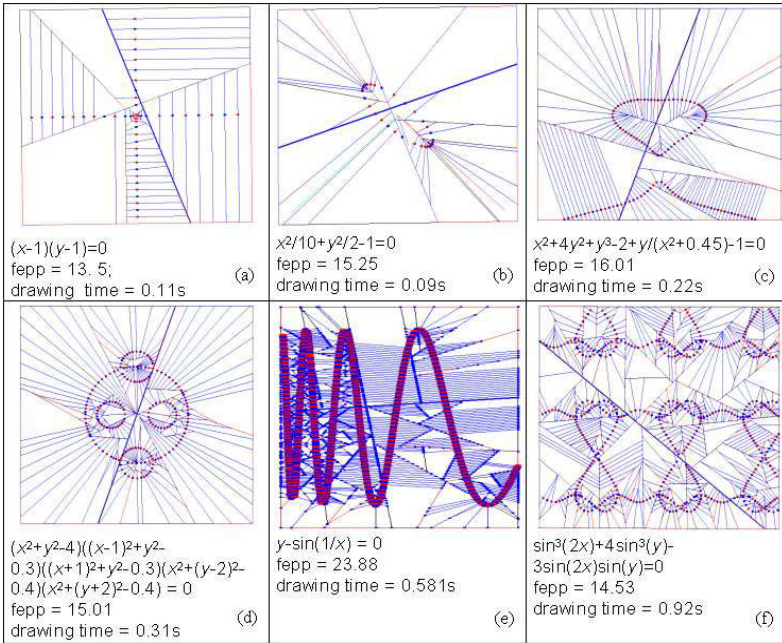


Fig. 5. Experimental results of some implicit curves.

recursion level of the BSP tree equal to 7. The term "fepp" denotes the number of function evaluations per computed point. This is so because every solution point is determined by an iterative approximation method (i.e. the secant method). The runtime performance tests were performed on a PC equipped with 500MHz Intel Pentium, 128MB RAM, and running Windows NT.

Commercial software packages such as the Maple and Mathematica do not incorporate accurate algorithms to draw implicit curves. In general, they are able to draw implicit curves somehow, but they fail at self-intersection points. For example, they cannot draw correctly the curve  $(x - 1)(y - 1) = 0$  in Fig. 5(a) about the point  $P = (1, 1)$ . In fact,  $P$  is a self-intersection point of this curve that is the union of two straight lines,  $x - 1 = 0$  and  $y - 1 = 0$ .

In general, space subdivision algorithms draw implicit curves correctly (see, for example, [10]). But, unlike the algorithm proposed in this paper, they are not able to distinguish a singularity from a regular point. However, similar to other space subdivision algorithms, our algorithm cannot identify isolated points.

## 5 Conclusions

A general implicit curve algorithm has been proposed. It is based on a non-uniform space partition technique. A curve needs not be closed or connected,

though single-point components are not detectable anyway. Besides, a curve may possess singularities and ripples.

Remarkably, this algorithm detects and accurately draws self-intersection and self-touching points without using any differential calculus tools. The computation of points is done through an iterative approximation technique, i.e. the secant method.

The algorithm proved to be fast enough to be included into current graphical systems. But, its speed depends on the efficiency of the algorithm that computes the intersection between a straight-line segment and the curve.

## References

1. Allgower, E. and Georg, K. *Numerical Continuation Methods: An Introduction*. Springer-Verlag, (1990)
2. Allgower, E., Gnutzmann, S.: *Simplicial Pivoting for Mesh Generation of Implicitly Defined Surfaces*. *Computer Aided Geometric Design*, Vol. 8, (1991) 305–325
3. Akai, Terrence J.: *Applied Numerical Methods for Engineers*. John Wiley & Sons Inc (1994)
4. Blinn, J.: *A Generalization of Algebraic Surface drawing*, *ACM Transactions on Graphics*. Vol. 1, No. 3, (1982) 235–256
5. Bloomenthal, J.: *Poligonisation of implicit surfaces*. *Computer Aided Geometric Design*, Vol. 5, (1988) 341–355
6. Bloomenthal, J.: *An Implicit Surface polygonizer*. *Graphics Gems, IV*, (1994)
7. Chandler, R.: *A tracking algorithm for implicitly defined curves*. *IEEE Computer Graphics & Applications*, Vol. 8, No. 2, (1988) 83–89
8. Keyser, J., Culver, T., Manocha, D., Krishnan, S.: *MAPC a library for efficient and exact manipulation of algebraic points and curves*. *Proceedings of the 15th ACM Symposium on Computational Geometry*, (1999) 360–369
9. Krishnan, S., Manocha, D.: *Numeric symbolic algorithms for evaluating one-dimensional algebraic sets*. *Proceedings of ACM Symposium and Algebraic Computation*, (1995) 59–67
10. Lopes, H., Oliveira, J.B., Figueiredo, L. H.: *Robust Adaptive Polygonal Approximation of Implicit Curves*. *Proceedings of SibGrapI* (2001)
11. Lorensen, W., Cline, W.: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. *Computer Graphics*, Vol. 21, No. 4, (1987) 163–169
12. Moller, T., Yagel, R.: *Efficient Rasterization of Implicit Functions*. (1995). (<http://citeseer.nj.nec.com/357413.html>)
13. Shewchuk, J.: *Adaptative precision floating-point arithmetic and fast robust geometric predicates*. *Discrete Computational Geometry*, Vol. 18, No. 3, (1997) 305–363
14. Snyder, J.: *Interval arithmetic for computer graphics*. *Proceedings of ACM Siggraph*, (1992) 121–130
15. Triquet, F., et al.: *Fast Polygonization of Implicit Surfaces*. *WSCG' 2001, Vol 2*. (2001) 283–290