A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Computer Science and Engineering
in the Graduate School of
the University of Aizu

Robust and adaptive polygonization of implicit curves and surfaces.

by

Kazuhiro Mochizuki

February 2004

The thesis titled

Robust and adaptive polygonization of implicit curves and surfaces.

by

Kazuhiro Mochizuki

is reviewed and approved by:

Main referee *Associate Professor* *Michel Cohen*	Date
Professor *Gennadiy Nikishkov*	Date
Assistant Professor *Roman Ďurikovič*	Date

The University of Aizu

February 2004

Contents

Acknowledgement

I wish to thank Dr. Michael Cohen for his support and encouragement. I wish to express my appreciation to Associate Professor Carl Vilbrandt and Dr. Alexander Pasko in Hosei University for his excellent advice and diligent efforts to guide me through this project.

I would like to thank the thesis reviewer, Dr. Roman Ďurikovič and Dr. Gennadiy Nikishkov for their valuable and constructive comments. I am grateful to Mr. Yuichiro Gotou in IT Research Institute at Kanazawa Institute of Technology for his advice and support on implementation. I would like also to thank Jody Vilbrandt for excellent support in writing this paper.

I would like to thank Mr. Juhn Yamadera and Dr. Michael Cohen again for giving me the opportunity to attend the academic conference, SIGGRAPH 2003. Some ideas in this paper are inspired in that conference.

I want to thank my friends in the Computer Arts Laboratory, especially Mr. Katuo Fujiwara and Mr. Turlif Vilbrandt, for many exciting discussions, and I would like to thank again Associate Professor Carl Vilbrandt and Dr. Alexander Pasko for their tramendous support in writing this paper.
Finally, I would like to thank professors and officers of the University of Aizu for providing an excellent research atmosphere at the university.

Abstract

This paper suggests new polygonization of implicit curve and surface based on octree decompostion methods. We get *Robust* and *Adaptive* polygonization with Affine Arithmetic calculation and Implicit Linear Interval Estimations or Extended Implicit Linear Interval Estimations. By our method, reduction of computation, decrease memory consumption and increase the polygonzation accuracy are achieved.

keywords: Affine arithmetic, decomposition method, Octree-based structure, Implicit Linear Interval Estimations, Adaptive polygonization, polygonization, crask problem.

Chapter 1

Introduction

For describing geometric objects, many types of geometric representation are used, but mostly used geometric representations are: *implicit representation* and *parametric representation*. This paper is mainly concerned with *implicit representation* and its visualization with polygonization.

Implicit representation models are widely used in engineering, computer graphics, and mathematics. *Implicit representation* is defined with a field function as described in the form:

$$f(x) = 0, \quad x \in R^n \tag{1.1}$$

For two dimensional objects, the above formula is rewritten as $f(x, y) = 0$, and for three dimensional objects as $f(x, y, z) = 0$. For explaining this category's benefit, we discuss this *implicit representation* formula compared with the other category, *parametric representation*. This formula may differ in appearance from the *parametric representation* formula. For example, the parametric and implicit expressions for a unit circle shown in the figure below differ in their properties, although they describe an identical shape.

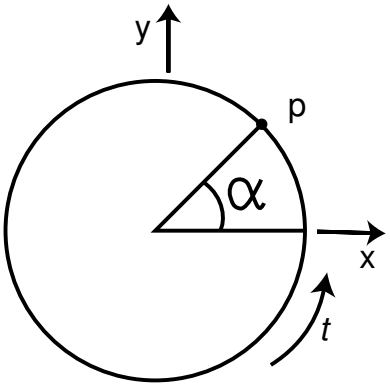

Figure 1.1: Simple circle.

Equiangular parametric function:

$$p = f(\alpha) = (cos(\alpha), sin(\alpha))$$
$$\alpha \in [0, 2\pi)$$

Non-equiangular parametric function:

$$p = f(t) = (\pm(1-t^2)/(1+t^2), 2t/(1+t^2))$$
$$t \in [-1, 1]$$

Implicit function:

$$p = f(x, y, z) = x^2 + y^2 - 1$$

From this example diagram, Figure 1.1, and the formula, the advantage and disadvantage of *implicit representation* may be understood. The main differences between *parametric representation* and *implicit representation* are the function argument types and their results. *Implicit representation* functions use coordinate arguments and yield level set values, which means the level set value of the cell's vertex vertex in relationship to th e implicit object where it is: inside (when $f(x, y, z) < 0$), outside (when $f(x, y, z) > 0$) or on the surface (when $f(x, y, z) = 0$). *Parametric representation* functions uses linear ratio, and the result is boundary coordinates. Therefore, *implicit representation* has an advantage in distinguishing any point as inside, outside or on the surface, but it is difficult to distinguish the surface and to manipulate local surface deformation. On the other hand, *parametric representation* has an advantage in visualizing an object, so most graphic applications use *parametric representation*. However, it is difficult to distinguish a point because of lack of information. For example, some extra calculation is required in morphing, blending or representing a soft object. Some researchers have already suggested Boolean operation with points [74][75], and in that application it is possible to use Boolean operation with *parametric function* in a simple model, but the idea is not used in a complex model. Therefore, *implicit representation* is gradually increasing in popularity in computer graphics, but visualization technique improvement is required.
In most applications, polygonization, or tessellation, is required for visualization. A polygon mesh is the one of the simplest forms of surface description and existing graphic systems, such as OpenGL, have special support for polygonal models, specially for triangular meshes. Although there is more sophisticated visualization description, such as B-splines or NURBS, polygonization algorithm is always required to describe three dimensional object. However, this task is complex for *implicit representation*. Previously, many researchers have tried to improve on such algorithms, and this thesis will also try to describe a solution for polygonizational visualization for *implicit representation*.

For implicit surfaces, any geometric model is defined by mathematical functions or discrete volumetric sets, eg. meta-ball or blobby object [7]. In this thesis, procedural mathematical functions, especially in the form of HyperFun models [46] are considered. Basically an implicit function is a mathematical method providing for the precise description of any object or any number of an object's measurable or imaginary physical attributes. However, the surface geometry as well as the other physical attributes of an implicit object are defined in a compressed form of mathematical notation that must be computationally unfolded before we can access and visualize the geometric information of an implicit object.
Consequently, the visualization of an implicit object requires a root or basic fundamental computational tool. A black box where in one side an implicit object is input and the other side of the black box outputs geometric information about the implicit object's surface needed for visualization providing for interactive modeling.

In this paper, we consider the adaptive polygonization for procedural mathematical implicit representation, and suggest one framework for polygonizing implicit functions in a robust way. In current methods used for polygonization, the total time of computational throughput and required memory are increased by a factor of eight in proportion

to an incremental increase in the accuracy of the surface information required for various applications.
Accordingly, this research proposes an adaptive polygonization procedure that predicts the implicit surface contour and wisely steps the procedural analysis of the geometry along the surface of the implicit object. This adaptive approach will provide for incremental increase in accuracy without the factor of eight increase in computational resources.

1.1 Problem Definition

In conventional polygonization framework, described in related work in chapter 2, memory requirement and total throughput is increased in proportion to required polygon fineness. The typical case, if user require doubled fineness in polygonization, exhaustive enumeration (Marching Cube [8]) calculation required octuple memories and calculation. On the other hand, continuation method [11] required quad memories, but they have some limitation that is not acceptable in discrete surface polygonization. So the problem of an octuple factor in computational resources of current conventional polygonization in proportion to any increase in geometrical fitness of the surface of an implicit object is critical to the general acceptance and application of implicit function. Thus, this is a basic problem that is worthy of research, even if the predictions of the continual doubling of computational resources is true.
For improving this problems, this paper suggest octree base polygonization framework. Our polygonization and thesis keywords are *Robust* and *Adaptive*. Both keywords relate to the polygonization problem in octree base decomposition polygonization.

Robust is related to the truncation problem described below. In this thesis, by Robust, we mean to guarantee cell detection whether the cell is intersected by an implicit object or not. The goal of this research is to guarantee the implementation of a robust and adaptive polygonization method that will produce the same quality of surface fitness as the current exhaustive enumeration method does with less total throughput.

- Truncation problem
  Each cell's level set value of vertex decides which cell is intersected by implicit surfaces. However, level set value of vertex information is ambiguous in deciding intersection. For example, sample cells below are never recognized as intersected cells. It is possible to detect such sample cells by decreasing cell size or dividing each cell into 4 sub-cells. However, this task would need to be done by the user manually, because the program does not know which cells really include a surface. Besides that, our polygonization framework uses decomposition method, and straddling detection in implicit curves is the most important criteria. So, this paper focuses on the straddling detection process.

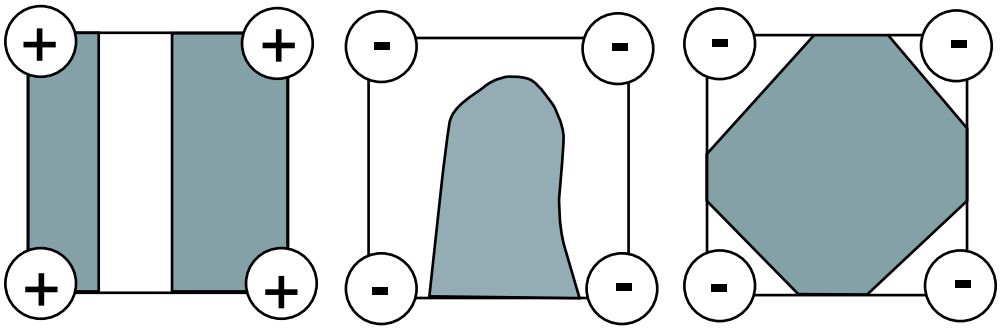

Figure 1.2: Ambiguity cells.

Adaptive means each cell fitness or level of local decomposition is controlled by local implicit surface curvature calculations based on the results of previous surface calculations.

1. Useless calculation problem
   Users are required by the polygonization program to set the cell size or dividing number of the bounding box, which is related with polygon size. In some cases, the user requires a quite fine surface, and extremely tiny cells are made in the cell creation step and, for each cell vertex, as a function is calculated; each cell is tested with its values whether it includes a surface or not. However, some tests are useless, because many parts do not include any surface. So, in the case of many fine polygons, most of the calculation is not directly related with the surface. In below diagram, a simple example is shown; left is the conventional way, and to the right is a better way. In some conventional solution, some extra computation is done because of no relation between cell decomposition calculation and each calculation result.

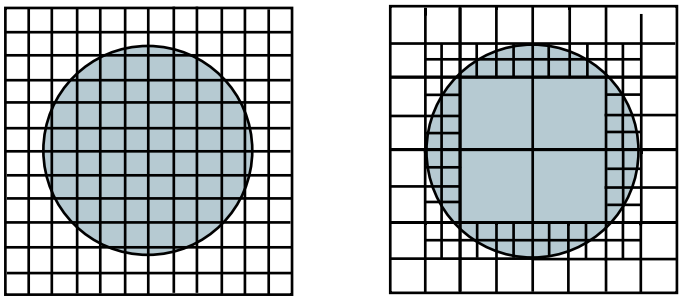

Figure 1.3: Cell size detection.

2. Adaptive computation problem
   At first, typically example in this problem is described. In conventional methods, a simple line is achieved as show in image on the left; the result is made from a collection of short lines. A more suitable representation of a straight line is the right image, one simple straight line. In some conventional solution, long line or plates is made by total segments, and there is no difference in the quality. Such

calculation is a waste and could happen in curves. The ideal solution is that tiny cells lie in high curvature area, and large cells lie in low curvature area.

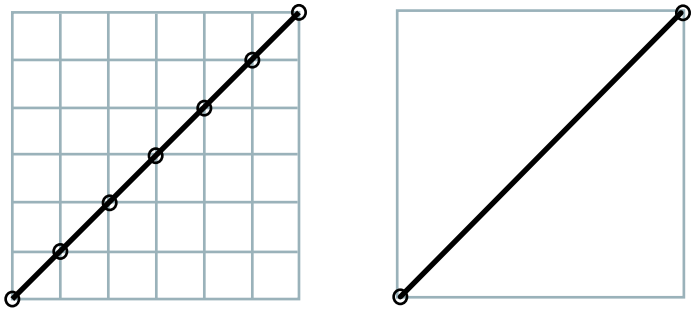

Figure 1.4: Simple line sample: y = x

The objective of our research is to suggest a more *Robust* and *Adaptive* polygonization which avoids the aforementioned problems with low computation compared with conventional technique.

1.2 Approach

To implement the above concept, we employ the hierarchical decomposition method and recursive subdivision method which we call Adaptive Decomposition Framework (ADF), with octree and quadtree data structure which describe procedural implicit functions. The hierarchical decomposition method is the basic framework for our polygonization program. Other techniques are uniquely gathered within the basic decomposition framework. The disparate techniques are used in a collaborative manner in order to achieve the goals of this research. The decomposition method with two collaborative techniques are listed below.

- Decomposition method

  Basic framework of our polygonization program. This method is selectively and recursively used to decompose the original cell into subcells. The following two techniques are used within this basic framework.

- Interval analysis

  This is used for analyzing the space defined by the bounding box, whether each cell includes an implicit surface or not. If cells are intersected and the cell size is not enough to describe the surface, the cells are subdivided into eight sub-cells and checked recursively until the cell is of a size that is sufficient to describe the implicit surface locally or where the curvature value in the cell is above the threshold, providing *Robustness* in surface identification.

- Curvature analysis

  This is the best criteria for *adaptive* polygonization. Conventionally, the curvature value is calculated from the derivative value of the implicit function, but we use another solution, Implicit Linear Interval Estimations. With this technique, a new unique *adaptive* calculation is achieved.

Interval analysis and Curvature analysis within the basic framework of the decomposition method constructs our main research approach using ADF to achieve *Robust* and *Adaptive* polygonization. The details of our approach are expanded as outlined in the thesis structure section.

1.3 Thesis Structure

The rest of the thesis is constructed as follows. Chapter 2 presents some conventional methods for visualization of implicit surface. Chapter 3 describes the basic structure and idea of hierarchical decomposition method, which is the basic framework for our polygonization method; the role of interval analysis and curve analysis are also mentioned. Chapter 4 discusses in detail the concept and calculation of the Interval Analysis techniques. Chapter 5 details Curvature Analysis solution with Implicit Linear Interval Estimations, and some calculation results, computation time comparison with conventional decomposition methods, and discussion with the reulsts are done in Chapter 6. Chapter 7 concludes this thesis and presents some future works.

Chapter 2

Related Work

In this section, some conventional methods for polygonizaiton of implicit curves and surfaces are described.
Generally, polygonization is classified into two steps, according to *Bloomenthal* [78]: *sampling* and *constructing*. *Sampling* means deciding points which may exist on the surface. *Constructing* means the creation of a data structure representing a polygonal approximation interpolating the sampled points. *Sampling* phase method is classified into two classes: *orderly sampling* and *unsteady sampling*. *Orderly sampling* is normally made from cells, as described in §2.1 below. Then, the *constructing* phase is done in each cell. Alternatively, *unsteady sampling* is directly related with *constructing* phase, because this method is designed for making appropriate points for constructing polygons, as described in section 3.2.

2.1 Spatial Partitioning Method

Spatial partitioning divides space into semi-disjoint cells which enclose the implicit surface. Each cell is utilized to make vertices which may be on the implicit surface. After that, the *construction* phase makes polygons from the vertices. Spatial partitioning is not only characteristic of polygonization, but for example, ray tracing algorithms use this scheme for detecting intersection of ray and surface. This idea is found in many references [15] [20] [41].

There are three principal types of partitioning methods: *exhaustive enumeration*, *continuation*, *subdivision*. By this classification, most polygonization is characterized. In the following sections, these three partitioning methods are described.

2.1.1 Exhaustive Enumeration

This type of spatial partioning is used in the well known conventional polygonization method, named *Marching Cube* [8], which was originally designed for making polygonal 3D models from medical data sets, such as computed tomography (CT) and magnetic resonance (MR). The algorithm based on hyperbolic arcs [13]can be used to automatically resolve the ambiguous cases.Usually, the cube is divided into a uni-

form grid, the number and distance of adjacent sample points are the same. In most of cases, a rectilinear uniform axis-aligned grid is utilized. After making cells, intersected cells are determined by enumeration examination between all cells, and those cells are polygonized. Because each cell's vertex is stored, the detection phase is very fast. For detecting the vertex position, linear interpolation is normally adopted.
A more detailed process of Marching cubes or the basic concept of exhaustive enumeration is described as follows:

1. Divide the box space for making cells with axis-allied grid.
   The box space, called *Bounding Box*, is divided into axis-aligned small cubic cells as shown in Fig. 2.1.

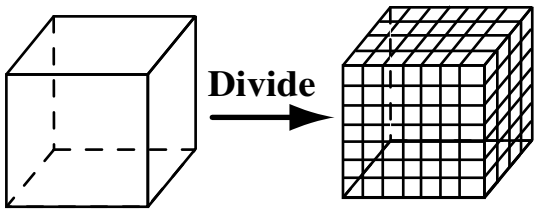

Figure 2.1: Divide bounding box.

2. Intersected cells are detected from the analysis of the cell's level set values of its four vertices. The three possible set values of a positive number, 0, or a negative number are obtained by the evaluation of the implicit object's function which defines it. If the set value is a positive number, the point is inside the implicit object. If the set value is zero, the point is on the surface of the implicit object, and with a negative set value, the point is outside of the implicit object. If the set value of the four vertices of one given cell are all positive or all negative, then the cell is fully contained within, or in the case of the latter, does not touch the implicit object. Therefore, logically, cells with all positive or all negative set values for their four vertices can not be intersected by the surface of the implicit object. Whereas intersected cells have both positive and negative set values for their vertices or a change in sign between their vertices.

3. In this phase, the point of intersection is achieved with linear interpolation between the vertex values of the cells vertices that have a change of sign between their level set values. The points of intersection are used to create polygons representing the implicit surface contained within each intersected cell detected with the method above. The intersection or polygonization representing the implicit surfaces intersecting a cell has only fourteen basic patterns or ways the cell can be intersected by the implicit surface. Thus most polygonizers use an index or look up table for the creation of the polygons to represent the implicit surface.

The above steps are the main concept in *Marching Cube*, and each of these processes has already been modified by some researchers. Hexagonal sampling, instead of unit sampling described in the first process, is suggested in [67][71]. To improve the speed of second process, a table may be used to polygonize cells [6]. Details concerning the generation of the index table are given in [11] and implemented in [78]. By this technique, efficiency of the second process is improved by 30% [57]. Hans Christian

Hege presented extended non-binary classification Marching Cube [38]. He improved the third step for non-binary polygonziation. By this extension, it is possible to make a more accurate triangulation with Boolean operations.

There have been so many improvements for Marching Cube, but an inherent weakness, efficiency, is not improved. For example, if the linear resolution of the grid is 100, then the total number of cells is $100^3 = 1,000,000$. Consequently, memory management becomes a significant burden. In that case, Jules Bloomental suggests using the *continuation method*, described in the next section.

2.1.2 Continuation Method

Incremental extension of surface polygons is the basic idea for the continuation method. This technique is usually divided into two classes: the predictor corrector (PC) and piecewise linear (PL) [78]. However, predictor correct method does not use cells, so in this section only piecewise linear is described, and the predictor corrector method is explained in a later section, ad hoc algorithms.

Piecewise Linear continuation method

Piecewise linear approximation has been applied to implicit surfaces using cubic cells [6][11] [33] and tetrahedras[69]. This solution is the basis of most polygonization methods. Space is subdivided into discrete sub-cells, and individual cells are polygonized only if an adjacent cell is already polygonized. A simple example in 2-dimensions is described below in Figure 2.2. The polygonization phase, which is done in each cell, is almost the same as the Marching Cube algorithm using a look-up table [78][57]. It is more efficient than the exhaustive enumeration method, because exhaustive enumeration only checks nearby intersected cells. In efficiency, this algorithm has an advantage but is lacking in robustness in detection of seed-cells. For a simple primitives, like torus, box, or sphere, it is not a heavy task, because simple primitive's surface is continuously. However for mathematical models as used in this paper, seed-cell detection becomes a difficult problem because there are no information about discrtete surface numbers in one cell.

For example, below Figure 2.3 shows five surfaces in one cell. In this case, the continuation method requires five seed points for making polygons. This task is complex because below surface is defined by $f(x, y, z) = x^2+y^2+z^2+xyz-0.5x^2y^2z^2-0.25$, and this formulation never yields required seed points number and discrete seed points. So, linear continuation method is not versatile.

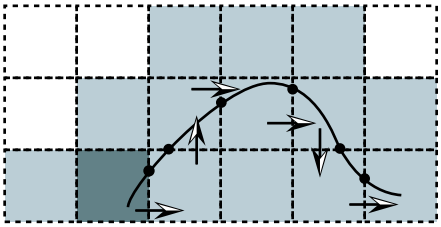

Figure 2.2: Piecewise linear method in contour drawing [78]

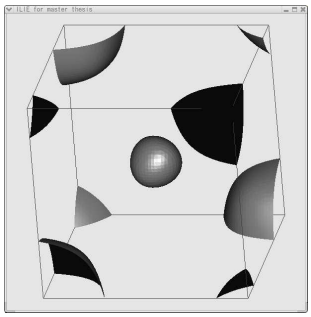

Figure 2.3: Discrete surface.

Dark gray represents seed-cell, light gray investigated cell, arrows describe propagation order, and white cells are never considered.

2.1.3 Subdivision Method

This algorithm is the recursive division of space into sub-cells that enclose the implicit surface [9] [17]. Our algorithm is classified into this group. Such calculation also yields a hierarchical data structure as an oct-tree of volume [31][40][55]. In three dimensions, subdivision of a cube yields same size eight cells which exist in the original subdivided cell's vertices, and subdivision proceeds until the terminal node cell's condition satisfies criteria, cell size, curvature and so on. In the last decade, many researchers tried to improve this algorithm.
This solution became popular after John M. Synder's and Tom Duff's theses [18][17]. In Duff thesis, a robust way is suggested with Interval Arithmetic (IA), which is a numerical solution technique proposed by R. E. Moore in 1966 [2][4]. At first this solution was used for error analysis [3] and maximum peak detection [5]. John M. Synder imported this idea for resolving some computer graphics problems, especially implicit surface polygonization in 1992 SIGGRAPH. His idea is quite simple, interval arithmetic is used for detecting non-intersected cell, called space pruning which was first derived by Devendra Kalra [15] with ray-tracing in implicit surface. After this paper, some researchers used this approach, ray-tracing [20] [35] [47] [76], comparison between Karla's model [15] and Tom Duff's model [17] [37], Boolean operation with interval arithmetic [36], parallel method [40]. IA has a problem of being over conservative, and Joao Luiz Dihl Comba proposed a substitute solution for interval arithmetic, called *Affine Arithmetic* [19]. This technique has also been extensively researched in polygonization [24] [26] [70], surface intersection [30], shaders [39], ray-casting [41], spherical coordinate polygonization [50], approximate parametric curve[72]. Some researchers purely describe Interval Arithmetic and Affine Arithmetic (AA). Adrian Bowyer's team [48] [64] described comparison between IA and AA in the following manner: they said AA is not perfectly superior to IA, but in most cases AA yields

tighter consequences, suggesting a more effective calculation with AA [59]. Huaho Show also compared AA to Bernstein Hull Methods for drawing contour in two dimensions, and they also conclude AA is superior in most cases [65].
Katja Buhler suggests dynamic compresssion of cells produced by Affine Arithmetic for ray-tracing method with implicit surface [53][66]. Besides interval analysis techinique, tree-based structure is also described in some papers; Luiz Velho suggest useful utilization of hierarchical data structure [44].

A simple example in a two-dimensional subdivision model, drawing contour, is presented below Fig. 2.3. This subdivision yields a quad-tree method.

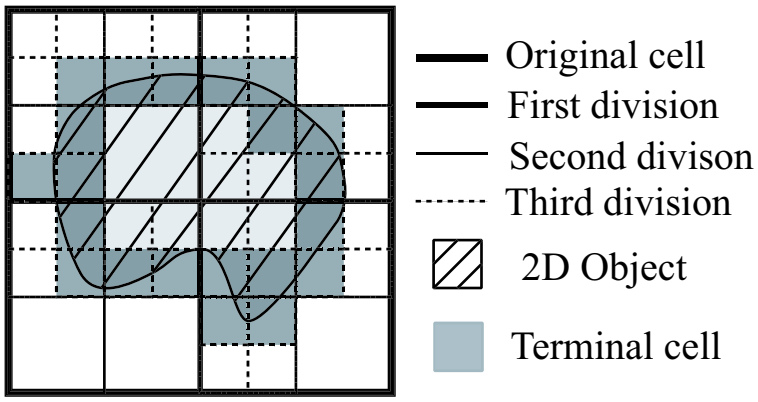

Figure 2.4: Subdivision method for contour drawing

As described in Approach in Chapter 1.2, our suggested methods are also classified into this method, and our methods is oriented in the extension and improvemnt [53][66] into polygonization.

2.2 Ad hoc polygonization

In this section, we describe some polygonization methods which amalgamates sampling and construction phases.

2.2.1 Shrinkwrap Polygonization

This approach builds a surrounding polygonized mesh and incrementally shrinks this mesh onto implicit surfaces [21][28]. This algorithm also suggests adaptive polygonization: relatively flat regions are approximated with large triangles, and tiny polygons exist on the small curvature area. One disadvantage of this algorithm is the difficulty in describing inner polygons. Therefore, we eliminate further discussion of this approach.

2.2.2 Predictor Corrector Continuation Method

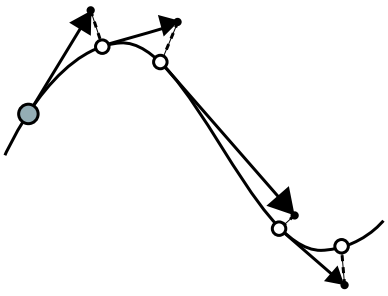

Figure 2.5: Predictor Corrector method in contour drawing.

Previously discussed piecewise linear continuation method extends surface polygons with extending cells. On the other hand, Predictor Continuation methods extends polygons by generating new vertices on the border of current polygonization [12][54]. The new point, p, is placed in the tangent plane of the active edge. A simple contour sample is described in Fig 2.4. In [54], the polygon is close to an equilateral triangle whose length has a correlation with the local curvature. The result looks fine, but such method also needs seed triangles in each closed model.

2.2.3 Particle-based Polygonzation

Unlike the above algorithms, a physically based particle polygonization method uses equilibrium configurations of simulated physical motions [16][23][29][58]. A sample result is shown in Figure 2.5.

In particle-based algorithm, it is possible to realize some merits, such as delauny-like triangulation property and adaptive triangulation. Delauny-like triangulation implies creation of nearly equilateral triangles; adapting triangulation implies adapting the particle density to locate surface curvature. However, a particle-based method has a critical draw-back, expensive computation. Tasso Karkanis compared particle-based system with his algorithm, continuation method [54], and also concludes it is an expensive computation task.

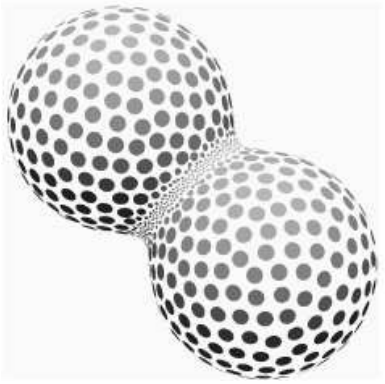

Figure 2.6: Particle based model [23].

2.3 Other rasterization solutions

The previous section mainly describes basic framework and overall approach to implicit function polygonization. In the following sections, we introduce more approaches which are adaptive and robust.

2.3.1 Adaptive Solution

The basic idea is that large polygons exist in low curvature areas and many small polygons exist in high curvature areas. One of the problems of adaptive polygonization is the cleft between fine and low curvature area; one solution that has already been proposed is the restriction of adjacent decomposition cell size [26][27][54][52].

2.3.2 Robust Solution

This solution has been a popular research topic during the last decade and is also the main focus of this research. This approach is used in cell detection to determine, which cell is intersected. Firstly, this idea is suggested by Karla and Alan H. Barr [15]. The first concept is quite differ from our approach, and the difference is described in next subsection. After that, Devendra Kalra suggested robust solution in implict function polygonization [15] with Interval Analysis [2][4]. After that thesis, this research topic is especially related with Subdivision Method. So, all papers described in Section 2.1.3 after 1992 SIGGRAPH is related with this solution. Especially, [48] [64][63] are focused on calculation throughput in mathmatical formulation calculation.

First Approach for Space Pruning

The first approach for robust Space Pruning with Interval Arithmetic was introduced by *Devendra Karla, and Alan H. Barr* [15]. That paper advocated the use of basic concepts in ray tracing for implicit surfaces. Besides techniques used in ray tracing, some excellent basic concepts in quad-tree , or oct-tree, data structure, and one analysis technique for cell detection (whether the cells are intersected by an implicit surface or not) are also introduced. The Space Pruning cell detection algorithm, may be a good solution in polygonization. In Karla's paper, this idea is used for achieving good performance in ray tracing. This paper uses a special type of implicit function called LG-surface description which has bounds on the net rate of change of the function and its directional derivative. Under this definition, L is used for the Lipschitz constant and is equal to or greater then the maximum rate of change f(x) in R. This L is used for detecting cells. Let X_0 be the center of the cell and d be half the length of the principal diagonal in the cell. Since the maximum rate of change with respect to distance of f(x) is L and maximum distance from X_0 is d, the maximum change in a cell is defined as $L \times d$. Hence if

$$|f(x_0)| > L \times d \tag{2.1}$$

then this cell is guaranteed to stay the same sign, and never assume a vlue of zero in this cell and hence this cell should be thrown away. For example, in Figure 2.7, (b) would be thrown away and (a) would seem to be straddling which is never decided for straddling surface by level set value of vertex based solution.

This technique resembles the technique proposed in this thesis in terms of value analysis, but there is a drawback in this method. In this method, L is required in the normal definition. This technique does not work well, because it is hard to get an exact L value since this method is a perfect black box. Therefore, this algorithm is not used

in our algorithm.

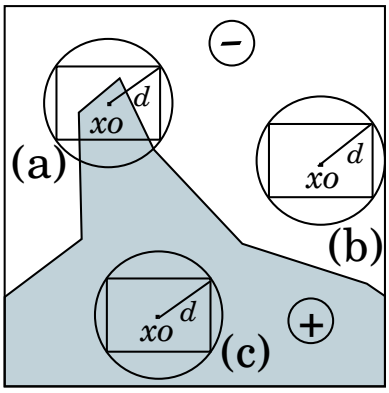

Figure 2.7: Sample category in [15].

Chapter 3

Polygonization Framework

In this chapter, we introduce the basic polygonization framework, ADF. We have already stated our polygonization category, *hierarchical decomposition method*, in Chapter 1 and Chapter 2, and we use Affine Arithmetic (AA) [19] and Implicit Linear Interval Estimations (ILIEs) [66] in our polygonization framework. For explaining the basic strategy with this two principles, AA and ILIEs, we introduce a more detailed procedure of our proposed polygonization, and we describe each algorithm's role in our polygonization framework.

3.1 Basic procedure in implicit surface visualization

At first, the basic procedure in our implicit surface visualization algorithm is outlined and classified in a list of four steps: preprocessing, dividing cell, vertex calculation, and building polygon. We describe each step's role in the visualization of an implicit surface.

1. Preprocessing step: making a function from an implicit object definition.
   Here, any implicit function is translated into a programming function. In our model, two types of functions are required: normal value argument function and the AA argment type function. In both function, the calculation notation is not different.

2. Dividing Cell step: making a terminal cell or an interval which straddles the implicit surface.
   This step is characteristic in the hierarchical decomposition method. Any implicit function defining the implict surface does not yield any information of thevertex position which lies on the surface. Therefore, the user needs to define the object space, called the bounding box, and the program checks in the bounding box. In the hierarchical decomposition method, the bounding box is recursively subdivided into sub-cells until some criteria is satisfied, and the resulting cell is yielded which may be straddling ithe mplicit surface; it is called the terminal cell. Our main concepts, *robust* and *adaptive*, are related here, and more detailed steps are mentioned in the following sections.

3. Vertex Calculation step: making vertex in each cell.
   Terminal cells, decided by the dividing cell step, may include a implicit surface. So, each cell's vertices are calculated and positions of the polygon vertices are estimated from the function's value. In our polygonization model, some improvement is also attempted in this step.

4. Building polygon (line) step.
   Polygons are created by connecting the vertices calculated in the previous step.

Our polygonization method implements the previous four steps, and in each step some improvement is carried out for achieving a *Robust* and *Adaptive* solution. These improvements are described in next sections.

3.2 Details of polygonization steps

In the sub-sections below, the four calculation steps, preprocessing, Dividing cell, Vertex calculation and Make polygon, are further elaborated focusing on our improvements and extension. In each of the steps, we describe in more detail their relationship with the *robust* and *adaptive* concept.

3.2.1 Preprocessing step

As mentioned above, any implicit object definition is needed to convert a program method in this step, and two types of functions are required for our method: normal argument types and Affine Arithmetic arguments types. AA arguments function is key to our polygonization framework. In our program, Affine Arithmetic is treated as a single parameter. Therefore, we only need to replace a normal value function parameter into an Affine Arithmetic parameter. In our test case, there is no difference between a double parameter function and an Affine Arithmetic parameter function. The example below is directly extracted from our sample C++ code, which defines a normal unit circle, and AAF, library value described in Chapter 6, indicates an Affine Arithmetic value. There is no difference excepting that the function parameter values are either AAF or double.

```
AAF function( AAF x, AAF y ){
    return 1.0*1.0 - x * x - y * y;
}

double function( double x, double y ){
    return 1.0*1.0 - x * x - y * y;
}
```

In our test case, any mathematical object is directly written into source code, and it is possible to extend this formulation into any implicit object parser, such as the HyperFun parser [46], because the basic calculation program code is not different from the normal one.

3.2.2 Dividing Cell step

In implicit surface or line visualization, cell dividing is a key point to visualization. Hierarchical decomposition method, or octree or quadtree based decomposition method, is used in our framework, and we use oct-tree data structure for maintaining hierarchical decomposition results. The fundamental step of hierarchical decomposition in dividing cell step, as described in Chapter 2, is recursive decomposition of the initial cell, called bounding box, into sub-cells. The recursive computation is not finished until fulfilling some criteria, width or curvature. The advantage of this type of computation is efficiency for calculation. It is possible to assemble the cells in the surface area for testing.
As described in Chapter 2, the key point of this type of polygonization is *robustness*. In the conventional solution, cell dividing is dependent on the level set value of vertex, and in that case the result can be ambiguous. For example, there is no unique solution to detect the case shown in Figure 1.2 as including a surface. This problem is avoided by the idea of Interval Analysis, as described by Tom Duff [17]. Detail computation and definition of Interval Analysis are described in the next chapter, so we only mention it here. Interval Analysis yields a range value for each cell. So, by employing this technique, a *robust* determination as to whether a cell is traversed or not is achieved. In our method, Affine Arithmetic is used for Interval Analysis, and the AA calculation yields the range of cell function.

The previous steps have been suggested in many other papers [19][24][26][30][39] [41][64]. We add some more improvements. First, we employ a cell pruning algorithm, which means getting rid of extra space before subdivision. Conventional decomposition polygonization uses regular interval decomposition; if the first cell is a regular grid cell, the terminal cell would be a regular cell. On the other hand, our terminal cells can be rectangles.

Second, we adopt curvature analysis which is related to the decomposition criteria. Some researchers have already suggested the same criteria for detecting cell size or polygon size [27][44][52]. Basically, all other techniques directly calculate curvature in points or interval. In our method, we do not calculate such curvature values. Instead of accurate curvature calculation, we estimate rough value in surface curvature.
Both added algorithms, cell pruning and curvature analysis, are achieved with Implicit Linear Interval Estimations [53][66]. Accordingly, each cell is compressed into a line range, as shown in Figure 3.1 below.

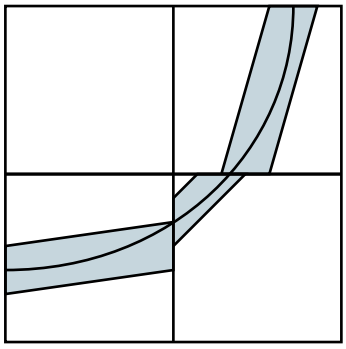

Figure 3.1: Divide bounding box.

In the above figure, the dark gray area expresses Implicit Linear Interval Estimations, which are derived from Affine Arithmetic result. This interval result is guaranteed to involve an implicit surface. So some computation is done with this idea. Pruning is done with ILIE's result, because a surface must exist in the dark gray area. Thus, there is no need to calculate in the extra area of the cell. For example, in Figure case 3.2, the lower right area is pruned.

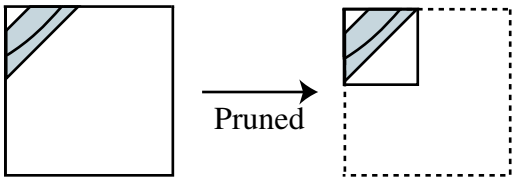

Figure 3.2: Pruning cell.

Additionally, the gray area width becomes one criteria for brief curvature value. In most previous polygonization techniques, accurate curvature value is used as a criteria for deciding polygon resolution level; curvature value is used for deciding more subdivisions or not in the cell pruning step, so accuracy of curvature is not so important for cell curvature analysis.
In such usage, width of ILIEs is enough for recursive decomposition. If the width of ILIEs is less than a tenth of the required cell size, its variation is not so different in the terminal cell or ILIE result yielded in the midstream step. Therefore, it is possible to be an alternative criteria for accurate curvature.
The above two concepts are extensions for conventional decomposition polygonization and make our polygonization an *adaptive* solution. Finally, pseudo programming code in dividing cell steps is shown below. Most of below code is the same as [66], excepting * line. So, all of the ILIEs described in later chapters do not implement pruning marked with * line, whereas EILIEs, described in 5.2.2, implement this point, because ILIEs denote simply conversion of [66] into polygonization, whereas EILIEs implement our extended implementation.

```
Algorithm decomposition( Interval interval ){

        // Affine arithmetic is done with interval
        if( check_intersection( interval) != TRUE ){
                return;
        }

        if( diameter(interval) < smallest width ){
                interval.terminal = TRUE;      //set interval terminal
                return;
        }

        ILIEs = calculate_ILIEs();

        if( diameter_ILIEs( ILIEs) < smallest width ) {
                interval.terminal = TRUE;      //set interval terminal
                return;
        }

        // Pruning Interval with ILIEs
        pruned_interval = pruning( interval );

        // Re-calculate ILIE
        ILIEs = calculate_ILIEs();

        // In 2 degree, subdivide 4 cells, In 3 degree, subdivide 8 cells.
        sub-interval = subdivide( pruned_interval into) ;


        for i = 1.... 8 {
                // Check each cell intersects ILIEs, or not
                if( ILIEs intersect sub-interval[i] ) {
                        // after pruning recursive decomposition
*                       decomposition( pruning(interval )  )                }
        }
}
```

Bold letters indicate improved points which are not implemented in the conventional polygonziation program, and last pruning is not implemented in ILIEs [66].

3.2.3 Vertex Calculation step

After the dividing cell step, terminal cells are obtained, and vertices are estimated in each terminal cell. In the conventional solution [26], functions are calculated in eight cell vertices, and vertex positions are estimated from the vertices calculation values with linear interpolation. In our solution, linear interpolation is also used to estimate

vertex position, but differs from the conventional way in the vertex position. In our solution, line or plate segments are yielded from ILIEs in the dividing cell step, so cross points between ILIEs and cell edges are used instead of eight cell corner. In Figure 3.3 below, each cell has four cross points with ILIEs area, denoted by dark gray, and functions are commutated in each vertices. That result is used for estimating vertex position, so consequently, accuracy of the surface vertex is increased, because our calculation position is narrower than conventional cell based interpolation. So, we get more accurate surface position in the same cell size than conventional method.
However, there is a disadvantage in this solution, called the *cracks problem*, as shown in the two dimensional example Figure 3.3 below:

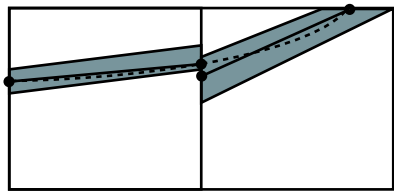

Figure 3.3: Cracks between vertices result.

All vertices are estimated with linear interpolation of each level set value of vertex, so if the width of the adjacent cell's ILIEs is different, the resulting coordinate differs in one edge as shown in Figure 3.3. In the above case, we suggest fitting to the narrow range vertex, because the exact vertex point is calculated with the narrow range.
Besides above crack problem, in three dimension adaptive decomposition, another crack occurs between different levels of cell resolution. This type of problem is common in adaptive decomposition. One example is described in Figure 3.4.

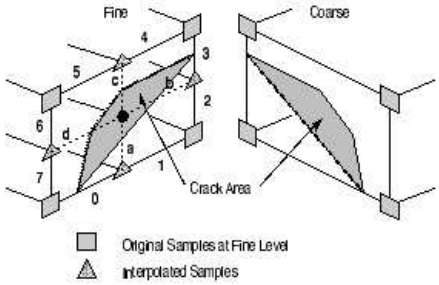

Figure 3.4: Cracks in three dimension [45].

Between different level of cell resolution, as Figure 3.4, cracks occur. Some researcher's solutions have already been suggested in [22][32][49]. However, these solutions are designed for only regular cell adaptive decompositions, and not easily used in our case because of pruning. In some cells computed by our method, cell size is different, and it is hard to find adjacent cells.
So, some improvement is needed to find adjacent cells, but it is possible to extend some reference ideas into our polygonization. For example, the simplest solution is fitting to coarse solution. In above Figure 3.4, fine cell vertex position is changed for fitting coarse line position. For adapting this simplest solution into our solution, one more fitting is needed. Because edges end position between fine and coarse are also different. So, edge vertices also need to fit narrow one, but it is possible to use the

simplest solution in our solution.

3.2.4 Building polygon (line) step

In this step, we explain how to make polygons from the estimated vertices. We suggest two solutions using exhaustive enumeration polygonization method [8][13][57] and dual contouring method[61][62][56]. It is possible to use both polygonization methods, but our application supports only exhaustive enumeration solution because of polygonization simplicity. Exhaustive polygonization steps are described below.

Polygonization based on cell vertices value analysis

This solution is the most cited solution for making polygons. The configuration of the set of polygons for a cubic cell is determined by the number of cell corners in which the function value is positive. Since each cell corner can take the binary state, in which the function value is positive or negative, there are 256 possible configuration for eight corners. The connectivities of cell edges for 256 possible configuration are stored in a look-up table, and 256 possible configurations are classified in fifteen basic configurations using rotation a diversion of the state in the eight corners. From this analysis, a look-up table with each corner vertices's value is implemented. With this look-up table, we get assortment in edge numbers. In the thesis by Gotou [57], each of the vertices, surfaces and edges in the cell are indexed as shown in Figure 3.5, and the combination for polygon creation is directed by the combination of edge numbers.

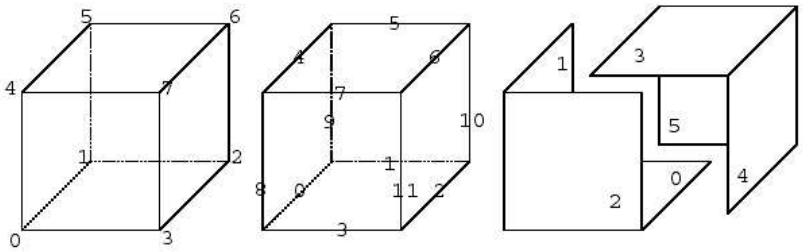

Figure 3.5: Cell indexing convention [57].

For example, if 0th and 1th corner have positive and others have negative values, 3(00000011) is used for the look-up table index. From this index, the look-up table yields combination of edges, $1 \rightarrow 9 \rightarrow 3$ and $3 \rightarrow 9 \rightarrow 8$. Thus, we use this look-up table for making edge vertices combination. To use this table, we need to detect each cell's vertices value, negative or positive. Fortunately, we have already checked each ILIE interval segment's vertices value. So, we utilize ILIE's vertices value for this look-up table value. In some cases, there is no corresponding vertex in the ILIE segments. For example, if the ILIE is straddling in edge 0, 8, 3, there are no corresponding vertices for 2, 5, 6 and 7. In that case, we set max or minimum value in vertex 4, 1 and 3 into no corresponding vertices value, because our implementation has already knows there is no possibility for straddling implicit curves. So, we do not need to calculate value in each vertex.

Chapter 4

Interval Analysis algorithm

In this section, we introduce one algorithm for detecting cells straddling the implicit surface, called *interval analysis*. With this algorithm, our framework can achieve *robust* solutions.

4.1 Interval Arithmetic

Robust methods for detecting cell straddling can be obtained by using range analysis, called *Interval Arithmetic* (IA). This solution can be found in R.E.Moore in 1966 [2]. IA automatically keeps track of rounding and truncation errors for each computed value. IA is widely appreciated as a highly efficient algorithm to manipulate uncertain data.
An early thesis with Interval Analysis in Computer Graphics, written by John M. Synder [18], suggests two algorithms, called SOLVE and MINIMIZE, to solve some computer graphics problems by using *Interval Analysis*, and one simple example of approximate implicit curves is shown. Subsequently, some researchers started to use IA in visualization of implicit function. In the next subsection, we introduce details of the IA technique.

4.1.1 Definition of Interval Arithmetic and Results

In Interval Arithmetic, quantities are represented by intervals and mathematical functions are extended to operate on intervals. In a more precise definition, a quantity $x \in R$ is represented in IA as :

$$x = [a, b], \text{ where a and b are real numbers(including } \pm\infty \text{)}$$
$$\text{with the understanding that } a \leq x \leq b$$

Arithmetic operations can be extended to intervals as [35]:

$$[a, b] + [c, d] = [a + c, b + d]$$
$$[a, b] - [c, d] = [a - d, b - c]$$
$$[a, b] \times [c, d] = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]$$
$$[a, b] \ / \ [c, d] = [a, b] \times [1/d, 1/c](provided \ \ 0 \ \notin [c, d])$$

$$x^n = \begin{cases} [b^n, a^n], \text{ if n even or } b \geq 0 \\ [a^n, b^n], \text{ if n odd or } a \leq 0 \\ [0, max(a,b)^n], \text{ if n even and } 0 \in x \end{cases}$$

With the above mathematical operation definition, Moore [2] showed that every computation function f has a natural interval extension function F, which include f results. Any algorithm for computing f can automatically be interpreted as an algorithm for computing F simply by composing interval formulas for the primitive operations. So, in the decomposition step, all implicit functions are extended into interval formulas, and cell intervals are used in parameters of that function: $X = [x\_lo, x\_hi], Y = [y\_lo, y\_hi], Z = [z\_lo, z\_hi]$.
Those extensions are specially powerful to implement with C++, because they support operator overloading, and in the example below we implement such an idea.
Figure 4.1 and Figure 4.2 are examples of IA's decomposition method in a two dimensional and a three dimensional visualization. Black boxes in two dimensions denote straddling cells, and vacant cells never straddle the implicit surface.

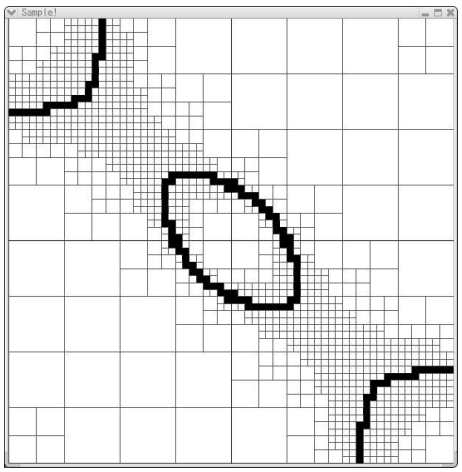

Figure 4.1: Interval Arithmetic result.

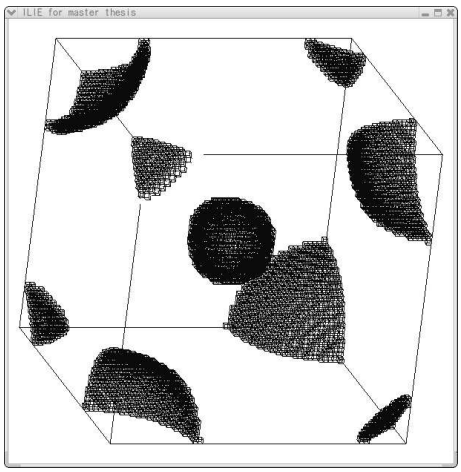

Figure 4.2: Interval Arithmetic result.

With this algorithm, *Robust* polygonization is achieved. However there is a problem to use this solution in visualization, called the *over-conservatism problem*.

4.1.2 Over-conservatism Problem in IA

Over-conservatism is caused by the Independence of the parameters used in definition in implicit function.

Simple sample of cover-conservatism is described in:

$$\begin{aligned} y &= x \times (4 - x) \\ x &\in [1, 3]. \end{aligned} \tag{4.1}$$

In the IA rule, the result range of the mathematical expression in [3.1] is calculated as

follows:

$$x = [1, 3]$$
$$4 - x = 4 - [1, 3] = [1, 3]$$
$$y = x \times (4 - x) = [1, 3] \times [1, 3] = [1, 9]$$

Real result is [3,4]. So, IA yields eight times wider result.

This over-conservatism in the IA calculation becomes a bottleneck in complex implicit functions, such as procedural implicit functions. Unfortunately, long computations are common for describing three dimensional objects. For solving this problem, two techniques have already been suggested by Joao Luiz Dihl Comba [19] and Irina Voiculescu [48], that is using Affine Arithmetic instead of Interval Arithmetic and formulation deformation. Formula deformation is using Bernstein-form and Horner-form instead of Power-form. As described in Interval Arithmetic explanation, Interval Arithmetic result is effected by calculation formulation. So, they focused on those calculation formulation. Some experiments of those formulation is done in [48], and comparisons are done in Martin paper [63]. In their comparison data, Affine Arithmetic is also described and recommended to use interval methods. So, basic concepts of Affine Arithmetic are described below, and we choose AA for our framework.

4.2 Affine Arithmetic

Affine Arithmetic (AA) is proposed in 1993 by Joao Luiz Dihl Comba [19]. The main difference between AA and IA is the error tracking style. IA range is represented by a numeric value. So there is no relation between source values and result values. On the other hand, AA is designed for considering correlations or dependencies between the sources and results, and it becomes an advantage in those calculation. By this advantage, AA produces a tighter range than interval arithmetic. We discuss the over-conservatism consideration with AA in section 4.2.5.

4.2.1 Definition of Affine Arithmetic

In AA, any variable is represented by the affine form in a first-degree polynomial. For example, the affine form of quantity x is represented as $\hat{x}$ and described as:

$$\hat{x} = x0 + x_1\epsilon_1 + x_2\epsilon_2 + ... + x_n\epsilon_n \tag{4.2}$$

In this definition, $\hat{x}$ is a known floating real coefficient, and ϵ_n is a fluctuation value and assumed to lie in the interval [-1,+1]. Each ϵ is perfectly independent value and x_n is a magnitude of each ϵ contribution.

In the Comba paper, each word is labeled such that x_0 is the *central value* of the affine form X; the coefficient x_i is the *partial deviations*, and ϵ_i is the *noise symbol*.

4.2.2 Conversion Between IA and AA

When using affine form to quantity x, the range of quantity x is represented with $\hat{x}$ as below:

$$[\hat{x}] = [x_0 - \epsilon, x_0 + \epsilon]$$
$$\epsilon = \sum \|x_i\| \tag{4.3}$$

This interval is computed with the assumption that each ϵ_i is mutually independent over [-1,+1]. The above definition is theoretical, so we introduce a more ordinary example to translate the interval definition into the affine form. Let's consider with interval $\bar{x}$ = [a, b]. We can convert this interval as in the formula below:

$$\hat{x} = x_0 + x_k \times \epsilon_k \tag{4.4}$$
$$x_0 = \frac{b+a}{2}, x_k = \frac{b-a}{2} \tag{4.5}$$

The noise symbol ϵ_k means the uncertain value width in the value of x, and ϵ_k = [-1, +1].

4.2.3 Affine Form Computation

In this section, the basic concept of Affine Arithmetic, which was introduced by [19], is described.
To evaluate an implicit formula with the affine form, we need to replace real value operations $z \leftarrow f(x, y)$ into affine form operations $\hat{z} \leftarrow \hat{f}(\hat{x}, \hat{y})$. $\hat{\text{f}}$ is a procedure for computing Affine Arithmetic in $z \leftarrow f(x, y)$. By definition, x and y is defined as follow:

$$x = x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n \tag{4.6}$$
$$y = y_0 + y_1\epsilon_1 + ... + y_n\epsilon_n \tag{4.7}$$

for some unknown value $\epsilon_1, ...\epsilon_n \in U^n$.

Using this changed converted value, z is defined with $epsilon_i$ as below:

$$z = f(x, y)$$
$$z = f(x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n, y_0 + y_1\epsilon_1 + ... + y_n\epsilon_n)$$
$$z = f^*(\epsilon_1, \epsilon_2, ..., \epsilon_n)$$

From the above definition, $f^*(\epsilon_1, \epsilon_2, ..., \epsilon_n)$ is replaced into the affine form:

$$z = z_0 + z_1\epsilon_1 + ... + z_n\epsilon_n$$

This formula perfectly fills the constraints described in (4.6) $\sim$ (4.7), and there are no more constraints in the above formulation. Besides this definition, we would like to describe mathematical operations in the next subsection.

4.2.4 Mathematical Operations with Affine Arithmetic.

We show each mathematical operation with Affine Arithmetic, as defined by previous papers [19] [48].

Affine Operation

The range of x and y is described by the definition as follows:

$$\begin{aligned} \hat{x} &= x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n \\ \hat{y} &= y_0 + y_1\epsilon_1 + ... + y_n\epsilon_n \end{aligned} \tag{4.8}$$

and $\alpha \in R$, then

$$\begin{aligned} \hat{x} \pm \hat{y} &= (x_0 \pm y_0) + (x_1 \pm y_1)\epsilon_1 + ... + (x_n \pm y_n)\epsilon_n \\ \alpha\hat{x} &= (\alpha x_0) + (\alpha x_1)\epsilon_1 + ... + x_n(\alpha\epsilon_n) \\ \hat{x} \pm \alpha &= (\alpha \pm x_[0]) + x_1\epsilon_1 + ... + x_n\epsilon_n \end{aligned} \tag{4.9}$$

Note that, according to the above formulas, the difference x̂ - x̂ between an affine form Dan itself is identically zero. In this case, the fact that the two operands share the same noise symbols with the same coefficients reveals that they are actually the same quantity, and not just two quantities that happen to have the same range of possible values [26]. This feature leads to some good computation performance, such as $(\hat{x} + \hat{y}) - \hat{x} = \hat{y}$ and $3\hat{x} - 2\hat{x} = \hat{x}$. These calculations are never done in Interval Arithmetic, and this is one of the reasons for effective analysis.

Non-Affine Operation

Commonly, f is not an affine operation, and it is difficult to express as in (4.9) formula, $f^*(\epsilon_1, ..\epsilon n) = f(\hat{x}, \hat{y})$. In that case, it is impossible to express ẑ into an accurate affine form with the combination of ϵ_i, so we would like to approximate f^* in the from:

$$f^a(\epsilon 1, ..\epsilon n) = z_0 + z_1\epsilon_1 + ... + z_n\epsilon_n \tag{4.10}$$

This formula is a reasonable approximation over domain U^n. The above information is not enough to express an exact value range, so $z_k\epsilon_k$ is adopted to express variation by a non-affine operation, and the resulting formula is expressed as below:

$$\begin{aligned} \hat{z} &= f^a(\epsilon_1, ...\epsilon_n) + z_k\epsilon_k \\ &= z_0 + z_1\epsilon_1 + ... + z_n\epsilon_n + z_k\epsilon_k \end{aligned} \tag{4.11}$$

ϵ_k is a brand-new and independent noise symbol produced by *f*, and z_k is a maximum value of $|f|$ in the given interval as below:

$$max\{|f^*(\epsilon_1, ...\epsilon_n) - f^a(\epsilon_1, ...\epsilon_n)| : \epsilon_1, ...\epsilon_n \in U\} \tag{4.12}$$

In the next step, the previous definitions are used to describe multiplication and square root arithmetic.

Multiplication and Square Root

In this section, we would like to consider about multiplication in the affine form. So, we would like to think about $z = f(x, y) = x \times y$. x and y are defined as (4.9), so ẑ represents quadratic a polynomial $f^*(\epsilon_1, ..\epsilon_n)$ as described below:

$$\begin{aligned} z &= f^*(\epsilon_1, ...\epsilon_n) \\ &= \hat{x} \cdot \hat{y} \\ &= (x_0 + \sum_{i=1}^{n} x_i\epsilon_i) \cdot (y_0 + \sum_{i=1}^{n} y_i\epsilon_i) \\ &= x_0y_0 + \sum_{i=1}^{n}(x_0y_i + y_0x_i)\epsilon_i + (\sum_{i=1}^{n} x_i\epsilon_i) \cdot (\sum_{i=1}^{n} y_i\epsilon_i) \end{aligned} \tag{4.13}$$

The best approximation of the affine operation in the above form into $f^*(\epsilon_i, ..\epsilon_n)$ consists of the formula:

$$f^*(\epsilon_i, ..\epsilon_n) = A(\epsilon_1, ..\epsilon_n) = x_0y_0 + \sum_{i=1}^{n}(x_0y_i + y_0x_i)\epsilon_i \tag{4.14}$$

Besides that, the last term is expressed:

$$\begin{aligned} Q(\epsilon_i, ..\epsilon_n) &= (\sum_{i=1}^{n} x_i\epsilon_i) \cdot (\sum_{i=1}^{n} y_i\epsilon_i) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_iy_i\epsilon_i\epsilon_j \end{aligned} \tag{4.15}$$

Q is the center-symmetric function and its domain U^n is also center-symmetric. From these properties, the affine approximation of $Q(\epsilon_i, ..\epsilon_n)$ is represented with a simple affine function, such as $Q = \alpha + \beta\epsilon_k$. So, if function Q's range is expressed by [a, b] over U^n, $Q(\epsilon_i, ..\epsilon_n)$ is translated as follow:

$$Q(\epsilon_i, ..\epsilon_n) \rightarrow \frac{a+b}{2} + \frac{a-b}{2}\epsilon_k \tag{4.16}$$

ϵ_k is a new noise symbol described in the above section. This formula satisfies $Q(\epsilon_i, ..\epsilon_n)$ range, but in most cases estimation of the precise max and min values in Q is a heavy task. Therefore Dihl Comba introduced one way for a quick conservative range estimation of Q.

$$\begin{aligned} &[-uv, +uv] \\ u = \sum_{i=1}^{n} |x_i|, v &= \sum_{i=1}^{n} |y_i| \end{aligned} \tag{4.17}$$

This definition reads $(b+a)/2 = (uv + (-uv))/2 = 0$, and $(b-a)/2 = (uv - (-uv))/2 = uv$; therefore, the multiplication form (4.15) is represented as follows:

$$\begin{aligned} \hat{z} &= \hat{x} \times \hat{y} \\ &= z_0 + z_1\epsilon_1 + ... + z_n\epsilon_n + z_k\epsilon_k \end{aligned} \tag{4.18}$$

$$\begin{aligned} z_0 &= x_0 y_0 \\ z_i &= x_0 y_i + y_0 x_i \\ z_k &= uv \end{aligned} \tag{4.19}$$

u and v is defined in (4.17). This interval is twice the exact range of Q, so some improvement is suggested for multiplication process. Two improvements for more precise multiplication are described in below.

One improvement for multiplication is suggested by Irina Voiculescu [48]. The basic concept of [48] is the power calculation in each noise symbol. Each AA multiplication yields new noise symbol, ϵ, as described in (4.19), and same transformation is done between same noise symbols, $\epsilon_1 \times \epsilon_1 \rightarrow \epsilon_2$. On the other hand, below multiplication is suggested in [48].

$$\hat{x}^a = (x_0 + x_1\epsilon_x)^a = x_0^a + \sum_{i=1}^{a} \left(\begin{array}{c} a \\ i \end{array} \right) x_0^{a-i} x_1^i \epsilon_x^i \tag{4.20}$$

Note that above polynomial (4.21), when i is odd ϵ_x^i will vary in the range [-1,1], and when i is even ϵ_x^i will vary in the range [0,1]. In other words, the noise ϵ_x^i varies over the same range in alternate term. For implement this idea, AA must have two noise symbol types, ϵ (=[-1,1]) and δ (=[0,1]).

Another improvement is suggested by Huahao Shou[59][63] which includes the power calculation, as described in [48], and the distribution law. For achieving this two concepts, they suggest using the Modified Matrix AA polynomial evaluation method (MAA). Basic MAA steps are described in Shou's thesis [59] and computation example is described in R.Martin [63]. MAA yields a tighter range, but computation is more complicated.
So, precise computation is done with more complex manipulation as suggested in above. The exact range of Q is computed in O($mlogm$), m means the total number of non-zero coefficients of the noise symbol in $\hat{x}$ and $\hat{y}$. It is difficult to decide whether performance or accuracy, because its criterion is complex. In this paper, two types of manipulation are tested, simplest one and the power calculation one.

The square root computation, $\hat{z} = \sqrt{\hat{x}}$, of the affine form $\hat{x} = x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n$ is described below [26]:

$$\begin{aligned} \hat{z} = z_0 + z_1\epsilon_1 + ... + z_n\epsilon_n + z_k\epsilon_k \\ z_0 = \alpha x_0 + \beta \\ z_i = \alpha x_i \\ z_k = \delta \end{aligned} \tag{4.21}$$

and each new constant number is defined as below:

$$\alpha = \frac{1}{\sqrt{a}+\sqrt{b}}$$
$$\beta = \frac{\sqrt{a}+\sqrt{b}}{8} + \frac{\sqrt{a}\sqrt{b}}{2(\sqrt{a}+\sqrt{b})}$$
$$\delta = \frac{(\sqrt{b}-\sqrt{a})^2}{8(\sqrt{a}+\sqrt{b})} \tag{4.22}$$

a and b is interval of [x̂], $x \in [a, b]$.

4.2.5 Extension for procedural function and mathematical function

In this section, we explain more mathematical functions, as sine, cosine, tangent and exponential function. Besides that, some conditional statement extension with Affine Arithmetic is also described.
At first, we explain each mathematical function with Affine Arithmetic. Unfortunately, it is impossible to take relevance between mathematical function result and inputed parameter. So, calculation is done after converting input Affine Arithmetic value into Interval Arithmetic as described in John M. Synder's paper [18]. We show an inclusion function for cos(f) with Interval Arithmetic value, [a, b] which is converted by input Affine value x.

$$cos([a,b]) = \begin{cases} [-1,1] & if \quad 1 + [\frac{a}{\pi}] \leq \frac{b}{\pi} \\ [-1, max(cos(a), cos(b))] & if \quad \left[\frac{a}{\pi}\right] \leq \frac{b}{\pi} and \left[\frac{a}{\pi}\right] mod 2 = 1 \\ [min(cos(a), cos(b)), 1] & if \quad \left[\frac{a}{\pi}\right] \leq \frac{b}{\pi} and \left[\frac{a}{\pi}\right] mod 2 = 0 \\ [min(cos(a), cos(b)), max(cos(a), cos(b))], & other situation \end{cases} \tag{4.23}$$

The result is converted into Affine Arithmetic again without no relation in input value's noise symbol. Similar functions can be constructed for operators such as sine, exponential and logarithm function.

Our implementation also consideres procedural function extension. In procedural description, some conditional statements function is used as *if*, *while* and *switch* statement. In assumption statement, affine value is not treated like one variable. More detail difference is described in the following simple example function by pseudo code.

```
Value assumption( Value in ){
    if( in < 0 ){
        return in;
    } else {
        return in + 2;
    }
}
```

In normal valuable calculation, this function is extremely simple. If input value is under 0, input value is simply returned, and otherwise returned with added two in input value. On the other hand, affine arithmetic, and also interval arithmetic, yields quite different situation. We manipulate above function with interval $x = [-1, 1] = \epsilon_1$.
This interval *x* applicable both situation. So, in interval case, each statement is calculated, and the result yields [-1, 0] and [2,3]. However, in normal function absolutely returns one value. Therefore, the above function with Interval Arithmetic yields [-1,3].
Affine Arithmetic is also calculated as Interval Arithmetic. The different point is that Affine Arithmetic needs to translate Interval Arithmetic format into Affine Arithmetic format.
So, some assumption computation with Interval Analysis is more complex than normal valuable analysis. We need to extend assumption idea into interval as to partitioning tool. In that case, current statement with Interval Analysis is not suitable because of the lack of some information making result interval.

4.2.6 Example of Affine Computation

To test the above formulas, we calculate one simple example, defined in the over-conservatism problem: $\hat{z} = x \times (4 - x)$ with $x \in [1, 3]$. The result of IA is [1,9], so AA yields much tighter results in this situation.

$$\begin{aligned} \hat{x} &= 2 + \epsilon_1 \\ 4 - \hat{x} &= 2 - \epsilon_1 \end{aligned} \tag{4.24}$$

From this result, formulation is calculated with simple multiplication rule as follows:

$$\begin{aligned} \hat{z} &= \hat{x}(4 - \hat{x}) \\ &= (2 + \epsilon_1)(2 - \epsilon_1) \\ &= 4 + 0 \times \epsilon_1 - \epsilon_2 \end{aligned} \tag{4.25}$$

Besides above result, we calculate with the power calculation Affine Arithmetic as follow:

$$\begin{aligned} \hat{z} &= \hat{x}(4 - \hat{x}) \\ &= 4 + 0 \times \epsilon_1 - \delta_1 \end{aligned} \tag{4.26}$$

From the above translation result, we get the Affine Arithmetic result, $\hat{z} = 4 \pm 1 = [3, 5]$, and the improved Affine Arithmetic result, $\hat{z} = 4 - [0, 1] = [3, 4]$. The simple Affine Arithmetic result is closer to the real result, [3,4], than the Interval Arithmetic result, [1,9]. Moreover, Affine Arithmetic with the power calculation yields the exact same result, [3,4].
This result shows the possibility to solve some over-conservatism problems, but further analysis between Affine Arithmetic and Interval Arithmetic is still needed. For example, in some simple power calculation, $f(x) = x^2, x = [0, 2]$, Interval Arithmetic yields more tighter and exact value, [0,4], than Affine Arithmetic value, [-2,4] or [-1,4].
Besides calculation precision, calculation complexity also needs to be discussed. Such analysis is described in the next section.

4.2.7 Comparison of Affine Arithmetic and Interval Arithmetic

We post two example results of AA and IA visualization in two dimensions and three dimensions, and the calculation times. All examples show the dividing cell step's result, and vertex calculation step is also done in each results. Therefore, the following computation times are directly related with the total computation time for the visualization application in implicit representation.
2D object formulation example from theses [26][53]:

$$f(x, y) = x^2 + y^2 + x \times y - 0.5 \times x^2 \times y^2 - 0.25$$
$(x, y) \in [-2, 2]^2$. Terminal cell size is 2^{-6}.

3D object formulation example:

$$f(x, y, z) = x^2 + y^2 + z^2 + x \times y \times z - 0.5 \times x^2 \times y^2 \times z^2 - 0.25$$
$(x, y, z) \in [-2, 2]^3$. Terminal cell size is 2^{-5}.

Computation time(second)		
	IA	AA
2D example	0.014561	0.017154
3D example	0.187885	0.340699

Computations are done with Pentium 4 2.4Ghz, 512M DDR main memory, and Geforce FX 5200 video card with 256MB DDR memory.

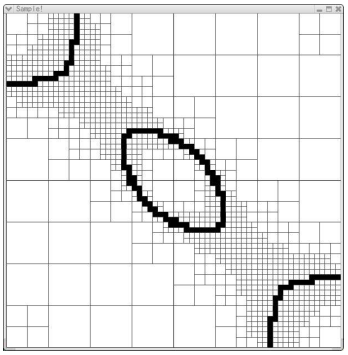

Figure 4.3: Interval Arithmetic result.

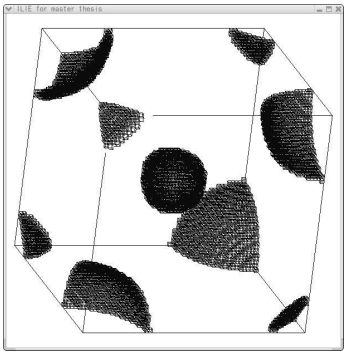

Figure 4.4: Interval Arithmetic result.

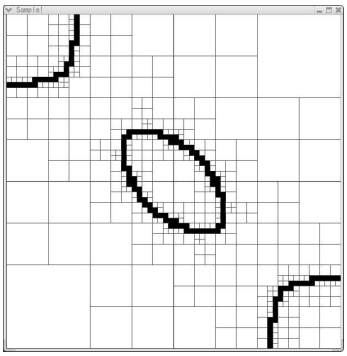

Figure 4.5: Affine Arithmetic result.

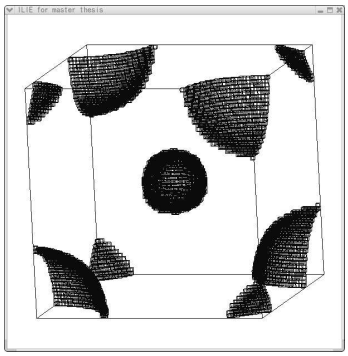

Figure 4.6: Affine Arithmetic result.

From the analysis in the 2D-example, some *over-conservatism problem* is avoided by using Affine Arithmetic, but basically the evaluation of implicit functions with Affine Arithmetic is a heavier task than with Interval Arithmetic. Especially in the three dimensional example, the calculation time is almost doubled, and the above examples are a typical case of the over-conservatism problem. On average, AA calculation time is almost 3-4 times slower than IA. Another researcher, Luiz Henrique de Figueiredo, said, "AA is typically 4-5 times slower than IA" [26]. This difference is not a trivial difference in computation. In the Figueiredo case, half size cells are yielded by IA in the same computation time.
The comparison of cell accuracy between IA and AA has already been checked by Affonso de Cusatis Junior [41]. In that work, AA does not give a good solution in all cases. In a few formulation cases, such as double torus and drop, IA gives better results, but generally AA is superior to IA in the terminal cell's performance.

From this conclusion, it would seem that we should choose IA, because some interval analysis with Affine Arithmetric advantage is overturned by the performance drawback. However, we choose Affine Arithmetic for our interval analysis solution, because AA has an important advantage for our purposes: ease of *Adaptive* calculation in polygonization. With *Adaptive* calculation, computation time will decrease below that of IA. Such computation is done with Implicit Linear Interval Estimations(ILIEs)[53][66]. The definition of ILIEs and *Adaptive* solution are discusssed in Chapter 5.

Chapter 5

Adaptive Calculation

Our polygonization framework adopts on *adaptive* analysis solution, and detailed steps of *adaptive* calculation are explained in this chapter. In the previous Chapter 4, we suggest using Affine Arithmetic for a *Robust* solution, and AA results are also used in our *adaptive* calculation steps. We have two computations for achieving *adaptive* polygonzation: cell pruning and criterion based on curvature in cell decomposition.

Before describing our algorithm in detail, conventional techniques for adaptive polygonnization are described for comparison, and after that, our implemented algorithm is explained.

5.1 Curvature Estimation in Conventional Way

Adaptive decomposition needs a criterion to decide the size of the polygon, and the curvature value is used to compute *adaptive* polygonization. To estimate how the curvature of the implicit surface varies in a cell, the plygonization program needs to compute and estimate the gradient of the implicit function inside a cell. There are three algorithms to estimate gradient: *finite differences approach*, *symbolic differentiation* and *automatic differentiation*.

5.1.1 The Finite Differences Approach

Finite Different approach is derived from the following gradient formula:

$$f^{'}(x) = \lim_{y \to x} \frac{f(x) - f(y)}{x - y}. \tag{5.1}$$

So, we get the curvature value to set boundlessly close value in parameters as a tanget of curve. This solution is attractive in its conceptual simplicity, and no additional computer programing is required. However, it is sometimes called the best avoided solution because of inaccuracy. The accuracy of a finite difference approximation depends upon the choice of delta, the input perturbation, and there is no a priori method for selecting data: too large and the evaluation will suffer truncation errors; too small and one is afflicted by round-off errors [10]. Therefore, researchers use this solution for curvature calculation in the decomposition step. Besides this solution,

symbolic differentiation is also suggested to get gradient in implicit surfaces. It manipulates an algebraic expression for f into an algebraic expression. This solution gets an exact range value in an implicit function, but it has disadvantage: it takes a long time to get the derivative expression. So, this solution is also avoided in some cases.
In visualization, especially in the criteria for the decomposition method, this solution is not appropriate because of accuracy and extensibility. We need to estimate certain range values in each cell, and it is difficult to estimate those values in this solution.

5.1.2 Automatic Differentiation

Instead of the above solutions, one ideal algorithm for deriving the function's gradient is suggested by [1][10], called Automatic Differentiation (AD). Derivative computation with AD is not an approximation value like the finite different approach: the only errors in AD evaluation are round-off errors, and these will be significant only when they already are significant for evaluating the function itself. So, this approach is used in some case, minimization problem [14]
Given a suitable primal code, AD is the process of automatically generating code that computes the gradient of the primal function. AD works by exploiting the fact that the chain rule can be used to evaluate the derivatives of a function no matter how complicated the primal function might at first appear. Once this is done, derivative are automatically computed for complicated expressions simply by following the rules for each mathematical elementally operation or function that appears in the evaluation of the function itself.

Helio Lopes polygonization use this idea for curvature value in adaptive calculation [52]. In Lopes paper, this technique is used with Intervals: all values in AD are considered as intervals; by this definition, the program can get interval estimates of partial derivatives automatically. From this computation, Lopes program achieved *adaptive* implicit curves with *robust* decomposition method, as shown in Figure 5.1.

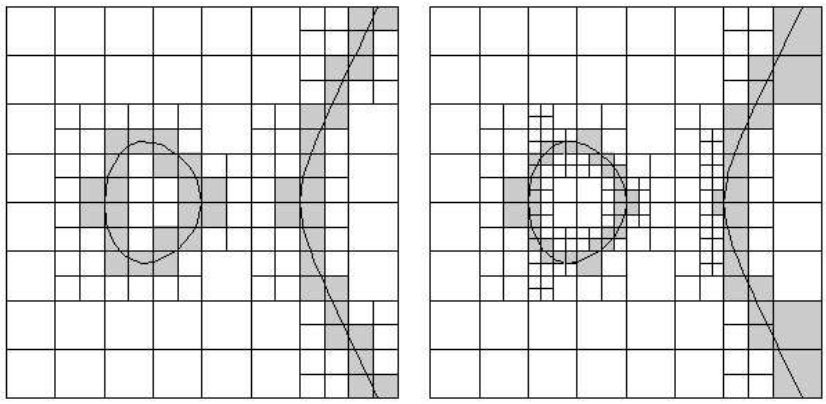

Figure 5.1: Adaptive Robust result in implicit curves quoted by Lopes's thesis.

As shown in Figure 5.1, Lopes's implicit curve polygonization program [52] achieves one of our suggested ideas in *Adaptive* solution: decomposition steps are stopped by curvature criteria. From the perspective of the resulting visualization, this solution perfectly achieves our demand, but this solution also has a disadvantage, reduces throughput. In the Lopes program, consequently, implicit function and automatic differentiation, need to be separately evaluated; computation throughput is doubled.

5.2 Adaptive solution in our method

Our approach is based on Implicit Linear Interval Estimations derived from Affine Arithmetic results, so extra computation is not required. This point is the advantage in our method.

5.2.1 Implicit Linear Interval Estimations (ILIEs)

The interval line or plate segment inside the axes-aligned box is called Implicit Linear Interval Estimation, and the ILIE segment is determined from the Affine Arithmetic calculation result. As described in previous Chapter 4, Affine Arithmetic is converted into an interval for detecting whether the result includes zero or not. In such calculations, some advantage of Affine Arithmetic is lost. To resolve this problem, Implicit Linear Interval Estimations is suggested by Katja Buhler [53][66]. In these paper, those segment is directly visualized by ray-tracing solution. In our approach, those segment is used cell detection steps.

The basic theorem of ILIEs is as follows:

- Implicit Object:
  F: $f(x) = 0$ in E^n.
  $f(x, y, z) = 0$ be an implicit surface of E^3.

- Interval:
  $\Pi_{i=1}^{n} \boldsymbol{I}_i \subset R^n$ a non-degenerated interval box.

- Each variables:
  $\hat{x}_i = \hat{x}_i(\epsilon_i) = x_i^0 + x_i^1 \epsilon_i$, $\epsilon_i \in [-1, 1]$,
  the corresponding affine form to interval $\boldsymbol{I}_i; i = 1, ..., n$.

- Affine Arithmetic result:
  $\gamma_j \in [-1, 1], \quad j = 1, .., m$

$$\begin{aligned} \hat{f}(\epsilon_1, ..., \epsilon_n, \gamma_1, ..., \gamma_m) &:= \hat{f}(\hat{x}) \\ &= f_0 + \sum_{i=1}^{n} f_i \epsilon_i + \sum_{j=1}^{m} g_j \gamma_j \end{aligned} \tag{5.2}$$

- ILIEs derived from Affine Arithmetic result:
  for $x \in \mathbb{I}_i$ and $\epsilon_i = \epsilon_i(x_i) := \frac{1}{x_i^1}(x_i - x_i^0)$, $i = 1, ..., n$

$$\begin{aligned} L(x) &:= \hat{f}(\epsilon_1(x_1), ..., \epsilon_n(x_n), [-1, 1], ..., [-1, 1]); \\ &= J + \sum_{i=1}^{n} f_i \frac{1}{x_i^1}(x_i - x_i^0) \end{aligned} \tag{5.3}$$

$$J := f_0 - \sum_{i=1}^{n} \frac{x_i^0}{x_i^1} f_i + [-\sum_{j=1}^{m} |g_j|, \sum_{j=1}^{m} |g_j|]n \tag{5.4}$$

From the above calculation, *Linear interval estimation* of F on $\mathbb{I}$ becomes:

$$L := \{x \in I | 0 \in L(x)\} \tag{5.5}$$

For explaining above formula meaning, one two dimensional sample example is described in here: $x = [0, 2]$, $y = [1, 3]$, $f(x, y) = y - x^2$.

$$\begin{aligned} \hat{x} &= 1 + 1\epsilon_x \\ \hat{y} &= 2 + 1\epsilon_y \end{aligned}$$

$$\begin{aligned} \hat{f}(\hat{x}, \hat{y}) &= 2 + 1\epsilon_y - (1 + 1\epsilon_x)^2 \\ &= 2 + 1\epsilon_y - 1 - 2\epsilon_x - \gamma_0 \qquad (\ \epsilon_x^2 \Rightarrow \gamma_0\ ) \\ &= 1 + 1\epsilon_y - 2\epsilon_x - \gamma_0 \\ &= -2\epsilon_x + 1\epsilon_y + (1 - \gamma_0) \end{aligned}$$

From this result, ILIEs is derived as follow:

$$\begin{aligned} L(x, y) &= -2\tfrac{x}{1} + 1\tfrac{y}{1} + (1 - \gamma_0) - (-2)\tfrac{1}{1} - 1\tfrac{2}{1} \\ &= -2x + y + (-\gamma_0 + 1) \\ &= -2x + y + J, \qquad J = [0, 2] \end{aligned}$$

In our implementation, iso-value becomes a surface. So, above ILIEs formulation is rewrited as below:

$$y = 2x - [0, 2] \tag{5.6}$$

ILIEs and original function are described in Figure 5.2 below:

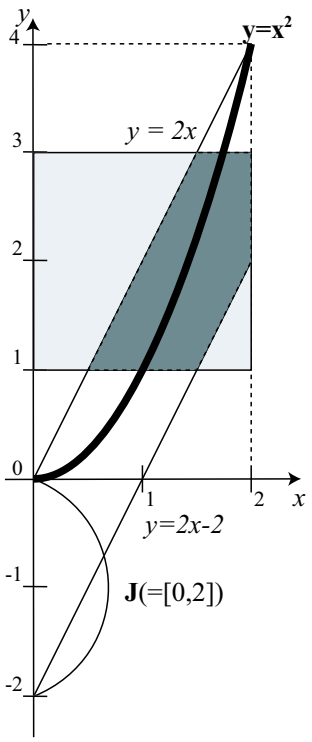

Figure 5.2: ILIE in one cell.

Light gray area in Figure 5.2 is corresponded interval area; $x, y = [0, 2], [1, 3]$, and dark gray area is corresponded to ILIEs segment in cell. As described in left figure, ILIEs provide linear, tight, simple and oriented enclosures for implicit objects inside a given cell.

Some advantages of ILIEs are:

- ILIEs is derived from Affine Arithmetic computation result, described in the above theorem. So, extra computation is not required to get ILIEs after computing Affine Arithmetic in a cell.
- The diameter of ILIEs, notated as **J**, is a good criteria to guess curvature in the area. From this information adaptive computation is done at an extremely low cost compared with the conventional curvature calculation method, because the conventional method required additional extra computation for estimating curvature in each cell and the calculation cost is almost the same as interval estimation [54].
- The terminal cell has ILIEs, providing a tighter piecewise linear interval which encloses an implicit surface.
- With ILIEs, unnecessary interval analysis of an implicit object is avoided without any extra complex computation. Described in Figure 5.4, after the original cell subdivision, each subdivided cell is tested as to whether the cell intersects, and only cells passing this test are more computed.
- ILIEs makes it possible for cell pruning before the decomposition step. After calculation of ILIEs, a portion of the cell is pruned, and the rest of the cell is subdivided. So, tighter cell subdivision is done.

5.2.2 Extended ILIEs

Besides that idea, our implementation extends a little in (5.4) formulation. As described in Chapter 4, Affine Arithmetic has some strategy in multiplication, and our framework can supports power formulation. Therefore, our affine arithmetic result is extended as:

$$\hat{f}(\hat{x}) = f^0 + \sum_{i=1}^{n} f^i \epsilon_i + \sum_{j=1}^{m} g^j \gamma_j + \sum_{k=1}^{l} h^k \delta_k, \qquad \epsilon_i, \gamma_j = [-1, 1], \delta_k = [0, 1]. \quad (5.7)$$

From this extension, (5.4) formulation is described as follow:

$$\delta_{min} = \sum_{k=1}^{h} h^k, \quad h^k = \begin{cases} d_k & (d_k < 0) \\ 0 & (d_k \geq 0) \end{cases}$$

$$\delta_{max} = \sum_{k=1}^{h} h^k, \quad h^k = \begin{cases} d_k & (d_k > 0) \\ 0 & (d_k \leq 0) \end{cases}$$

$$J := f^0 - \sum_{i=1}^{n} \frac{x_i^0}{x_i^1} f^i + [-\sum_{j=1}^{m} |g^j| + d_{min}, \sum_{j=1}^{m} |g^j| + d_{max}] \quad (5.8)$$

From this extensions, ILIE segment width becomes a little tighter than conventional solution, but there calculation complex is increased. Before discussing about the difference between original ILIE and extension ILIE, basic advantages of ILIE are discussed as follow, and the analysis in this extension should be discussed with some result in next section, because our extension is kinds of trade off in throughput and accuracy.

5.2.3 Curvature analysis with ILIEs

As described in the above ILIE advantage, it is possible to estimate ILIE width for estimating cell curvature. To describe this idea, we show one simple two-dimensional example:

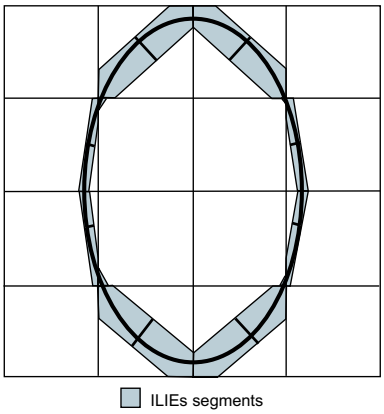

Figure 5.3: ILIE with ellipse in 16 cells.

In Figure 5.3, the dark gray represents ILIE area, and the black curved line represents an implicit line. This image shows that it is possible to use the ILIE width as a substitute for the real curvature value of an implicit object. In Figure 5.3 example, each upper and bottom two cells contains obviously wider range compared with other ILIEs segments. So, those cells is needed to subdivide one more times. In our implementation, we derive the ILIE segment width limit from the required cell size, tenth or fifth of required cell size.

As a matter of fact, this width value is
superior to the real curvature range value.
If we polygonize a wavy line, like a sine curve, in two degrees with extremely small amplitude and range, less than a tenth of the required cell size, ILIEs would stop the decomposition polygonization. On the other hand, real curvature value polygonization as described in automatic differentiation [52] should not stop decomposition steps until satisfying required cell size, because the wavy line curvature value has a wider range than normal value. This is a trade-off between accuracy and throughput, and ILIE can detect the degree with intuitive width. This is one reason for using ILIE in *adaptive* calculation.

5.2.4 Cell Pruning

As described in chapter 3, we use the cell pruning idea with ILIE. From the ILIE result formulation 5.3, we can get two segment formulation. For example, we get the following two segments in three dimensions:

$$J = [J_{lo}, J_{hi}]$$

$$L(x, y, z) = f^x \frac{1}{x_1} x + f^y \frac{1}{y_1} y + f^z \frac{1}{z_1} z + J_{lo} \tag{5.9}$$

$$L(x, y, z) = f^x \frac{1}{x_1} x + f^y \frac{1}{y_1} y + f^z \frac{1}{z_1} z + J_{hi} \tag{5.10}$$

We calculate the crossover area with the above two segments and the cell, and pruning is done as shown in Figure 3.2.

5.2.5 Intersection tests

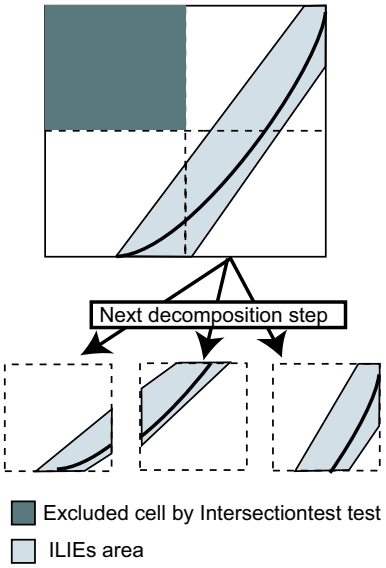

Figure 5.4: ILIE and intersection test .

Intersection tests are also a kind of *Adaptive* solution, and some extra computation is avoided without some extra calculation.
In our polygonization, the cell is divided into sub cells, and we use ILIE for detecting whether those subdivided cells are straddling the implicit surface or not before checking each cell value with Affine Arithmetic. ILIE is guaranteed to exist for implicit surface in those intervals, so this test is also *Robust*. The detail calculation is expalined with left Figure 5.4. One big cell is subdivided into four cells, and original cell has ILIEs area expressed with light gray in left Figure, and each subdivied-cells are checked for stradling ILIEs segmets, and if subdived cell is not straddling, decomposition calculation is stoped in that cell. With this test,

total throughput in our polygonization is increased, because normal computation of the straddling test with Affine Arithmetic is more computationally intensive than this crossing test.

We suggest cell pruning method after cell intersection test. By this extension, cell is more adaptive for the original lines.

That concludes our *adaptive* solutions, and we implement all suggested solutions in our visualizations. The main advantage of our *adaptive* solution is throughput. All *adaptive* solutions are achieved with extremely computation compared with conventional solutions.

The difference, conceptual advantages and disadvantage, in our suggested method are described in below with comparing Lopes's solution [52].

- Adaptive cell decomposition.
  Lopes's solution and our solution achieve to stop cell decomposition step with curvature analysis. The difference of our method is accuracy of estimated points. Lopes's method never yields segments like ILIEs. So, points are estimated by linear interval estimations with big cell edge's vertices. On the other hand, out method yields enough small segments compared with required cell size, and absolutely, more accurate vertices positions are yielded.

- Throughput.
  Lopes's solution used Interval Arithmetic for each cell calculation, in Automatic Differentiation and Interval Analysis. On the other hand, we use Affine Arithmetic in Interval Analysis, and cell decomposition criteria is derived from Interval Analysis result. So, our solution is not calculate twice as Lopes's solution. However, it is difficult to conclude which one is superior to another from throughput point, because Affine Arithmetic is more complex than Interval Arithmetic calculation as described in Chapter 4, but Affine Arithmetic generally yields tighter range than Interval Arithmetic and those range effects for total checked number. Therefore, it is not simplicity for generally conclude in throughput point.

- Gradient calculation.
  Lopes' solution makes Automatic Differentiation before cell decomposition step. Automatic Differentiation is not versatility in procedurally defined implicit function, but it yields exact gradient in any points. It leads good polygon optimization steps. So, this is one advantage in Lopes's solution which is not supported by our solution.

Further discussion is done in the next chapter with some two- and three-dimensional application results.

Chapter 6

Results and Discussion

This section compares the results of using conventional cell based polygonization methods: Marching Cube (MC), Interval Arithmetic (IA), and Affine Arithmetic (AA), with our Adaptive Decomposition Framework(ADF). ADF, as first describe in Section 3.3.2, implements all of the basic constructs of conventional cell based polygonization methods under an adaptive cell decomposition framework using the C++ language. We use, modify and extend the AA library *libaa* [77] for adaptive calculation. For example, we extend the AA multiplication as described in Section 5.2.2. However, the IA computational algorithms, the decomposition framework, and the attending decomposition data structures needed to support the framework are all made from scratch.

6.1 Result of Two-Dimensional Polygonization

At first, we show some dividing cell results with some required fineness for comparing, Affine Arithmetic, Implicit Linear Interval Estimations, and Extended Implicit Linear Interval Estimations. In cell graph, black area describing implicit line maybe straddling in that area. Dark gray area in ILIEs and Extended ILIEs implies cell are which includes ILIEs segments.
After comparison figure, three results in Marching Cube (MC), Interval Arithmetic decomposition (IA), Affine Arithmetic decomposition (AA), ILIEs decomposition (ILIEs), Extended ILIEs (EILIEs) decomposition is descriptives with below listed data.

- Detailed tables of computation times with second.
  Calculation time in two dimension is measured in total of dividing cell, vertex calculation and make polygon steps. That time is statistics with Pentium 2.4Ghz, 512M DDR main memory, and Geforce FX 5200 video card with 256MB DDR memory.

- Tables of number of terminal cells and total cells.
  Terminal cell means last cell which may be straddling the implicit curves analyzed by Interval Arithmetic or Affine Arithmetic. In ILIE situation, some curvature based results are also included into this area.

- Two line graphs of time and the percentages of terminal cell number in total cells.

The aim for making Percentage of straddling cells in total cells is indicating percentage of vain calculation which is not directly related for estimating line detection.

- Five dividing cell results with setting 0.0625 (2^{-5}) cell size (Marching cube, Interval Arithmetic, Affine Arithmetic, Implicit Linear Interval Estimations, and Extended Implicit Linear Interval Estimations).

- Polgonization results with EILIEs, ILIEs, and Marching Cubes by above cell decomposition results.

In the three detailed examples, from page 43 to 47, we show five results in MC, IA, AA, ILIEs, EILIEs cell division results with tablas of number of terminal cells and total cells, and Line graph of calculation time.
All of the two dimensional examples are calculated with the interval [-2,2] and criteria for ILIEs and ELIEs adaptive calculation is tenth of required width.
teste

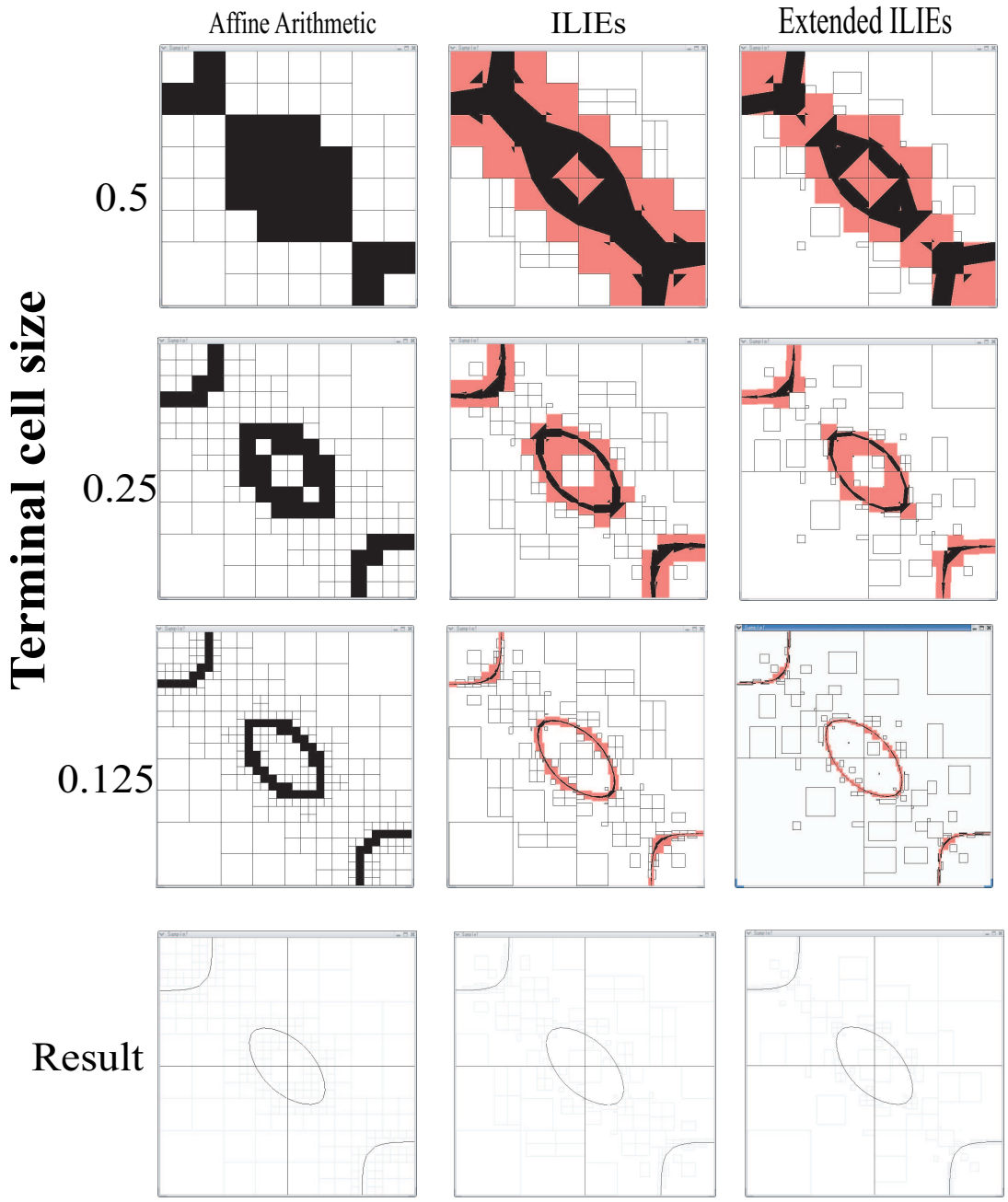

Figure 6.1: Fineness comparison of Affine Arithmetic, ILIEs and Extended ILIEs cell dividing steps.

Two dimensional example from implicit line thesis [52][53]:
$x^2 + y^2 + xy - 0.5x^2y^2 - 0.25$.
The bottom chart shows discrete values with computation times, and the following pages show image results with each method.

Time table (Second)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-2} (0.25)	0.000241	0.000982	0.002673	0.003764	0.005548
2^{-3} (0.125)	0.000725	0.002452	0.005327	0.007734	0.011188
2^{-4} (0.0625)	0.002651	0.006311	0.009069	0.013570	0.013794
2^{-5} (0.03125)	0.010583	0.014561	0.017154	0.015699	0.018789
2^{-6} (0.015625)	0.044242	0.027960	0.033129	0.022708	0.024520
2^{-7} (0.0078125)	0.183965	0.062654	0.067965	0.033282	0.035052
2^{-8} (0.00390625)	0.768375	0.131197	0.131458	0.045557	0.041247
2^{-9} (0.001953125)	58.256225	0.271288	0.257936	0.059808	0.065536
2^{-10} (0.0009765625	*	0.450151	0.44654	0.072971	0.075981

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-2}	38 / 256 (2^8)	38 / 172	38 / 160	52 / 130	48 / 106
2^{-3}	66 / 1024 (2^{10})	66 / 454	66 / 316	94 / 256	90 / 232
2^{-4}	134 / 4096 (2^{12})	134 / 1192	134 / 538	192 / 424	122/280
2^{-5}	266 / 16384 (2^{14})	266 / 2614	266 / 970	254 / 526	182/382
2^{-6}	522 / 65536 (2^{16})	522 / 5368	522 / 1816	382 / 736	254/496
2^{-7}	1038 / 262144 (2^{18})	1038 / 10834	1038 / 3442	518 / 994	390/703
2^{-8}	2074 / 1048576 (2^{20})	2074 / 21394	2074 / 6616	766 / 1390	456 / 802
2^{-9}	4146 / 4194304 (2^{22})	4146 / 42748	4146 / 12886	1050 / 1858	742 / 1258
2^{-10}	8282 / 16777216 (2^{24})	8282 / 85138	8282 / 25372	1398 / 2476	896/ 1492

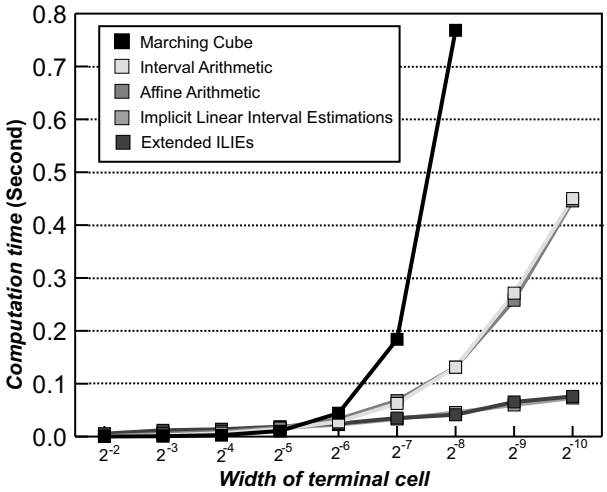

Figure 6.2: Calculation time with each polygonization result.

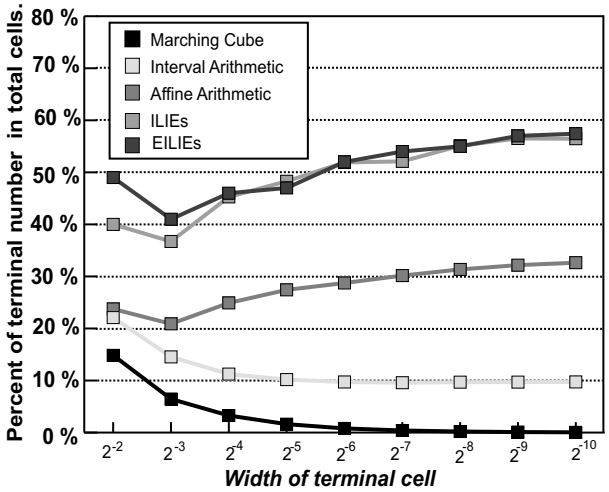

Figure 6.3: Percentage of straddling cells in total cell.

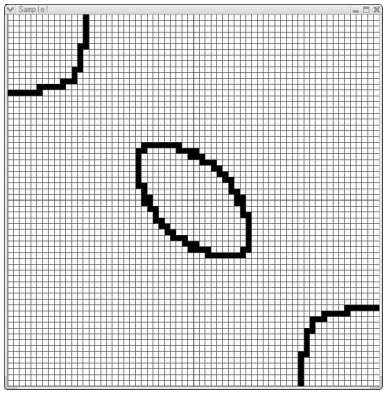

Figure 6.4: Marching Cube.

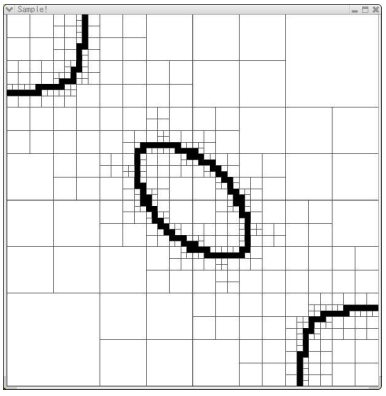

Figure 6.5: Affine Arithmetic.

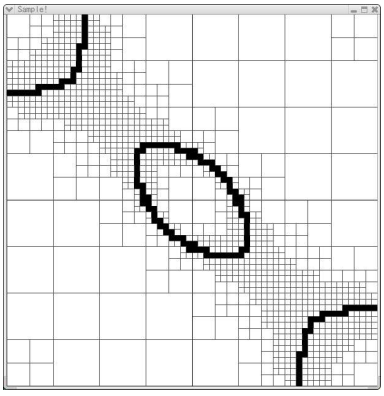

Figure 6.6: Interval Arithmetic.

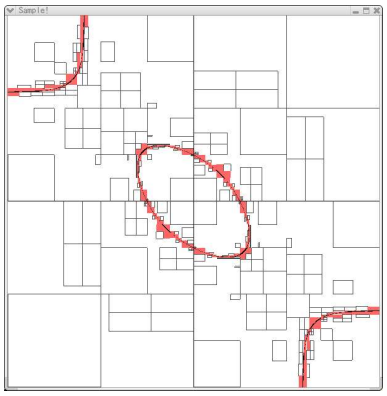

Figure 6.7: ILIE.

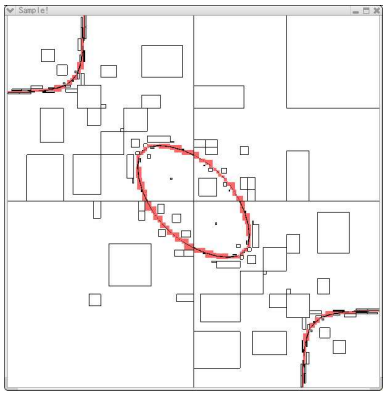

Figure 6.8: EILIE.

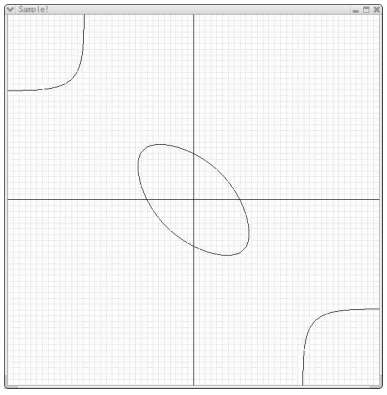

Figure 6.9: Marching Cuble rline.

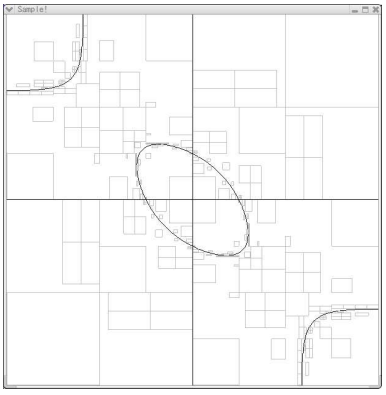

Figure 6.10: ILIE line.

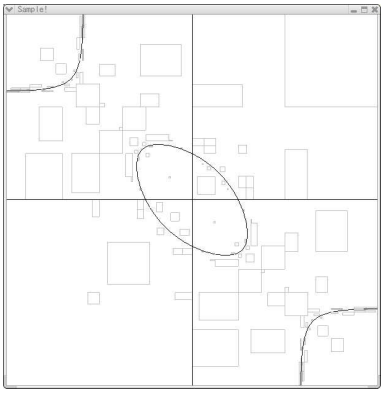

Figure 6.11: EILIE line.

Two dimensional example [52]: $y^2 - x^3 + x$.

The bottom chart shows discrete values with computation times, and the following pages shows image results with each method.

Time table (Second)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-2} (0.25)	0.000230	0.000682	0.001489	0.002712	0.003025
2^{-3} (0.125)	0.000740	0.001370	0.002992	0.003755	0.004247
2^{-4} (0.0625)	0.002740	0.002896	0.005834	0.005802	0.005490
2^{-5} (0.03125)	0.010407	0.005880	0.011080	0.007546	0.007037
2^{-6} (0.015625)	0.040972	0.012305	0.021878	0.009428	0.009308
2^{-7} (0.0078125)	0.164625	0.024708	0.042305	0.015433	0.012555
2^{-8} (0.00390625)	0.640320	0.049740	0.086855	0.019330	0.018360
2^{-9} (0.001953125)	28.563134	0.104857	0.170219	0.026459	0.022733
2^{-10} (0.0009765625)	*	0.204026	0.338537	0.036050	0.031276
2^{-11} (0.00048828125)	*	0.417024	0.671857	0.055172	0.044383

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	ELIEs
2^{-2}	44 / 256 (2^8)	44 / 136	44 / 136	58 / 130	40 / 82
2^{-3}	82 / 1024 (2^{10})	82 / 304	82 / 280	90 / 178	62 / 118
2^{-4}	162 / 4096 (2^{12})	162 / 658	162 / 556	140 / 280	78 / 148
2^{-5}	324 / 16384 (2^{14})	324 / 1360	324 / 1060	182 / 358	110 / 196
2^{-6}	646 / 65536 (2^{16})	646 / 2806	646 / 2062	234 / 442	148 / 250
2^{-7}	1292 / 262144 (2^{18})	1292 / 5674	1292 / 4030	408 / 706	209 / 340
2^{-8}	2580 / 1048576 (2^{20})	2580 / 11404	2580 / 7936	530 / 892	283 / 454
2^{-9}	5160 / 4194304 (2^{22})	5160 / 23014	5160 / 15694	732 / 1210	391 / 622
2^{-10}	10320 / 16777216 (2^{24})	10320 / 46132	10320 / 31198	1006 / 1642	541 / 844

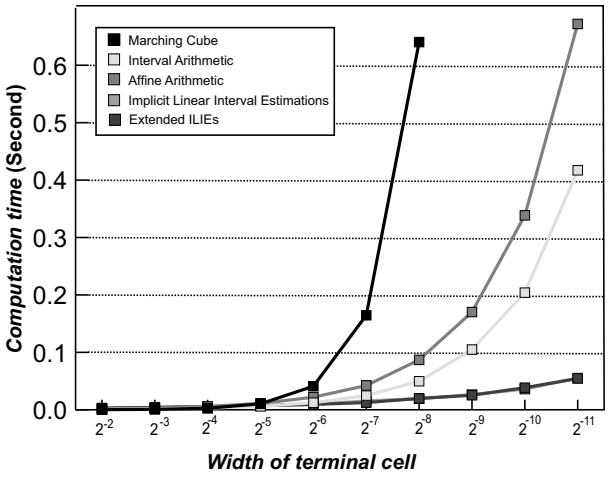

Figure 6.12: Calculation time with each polygonization result.

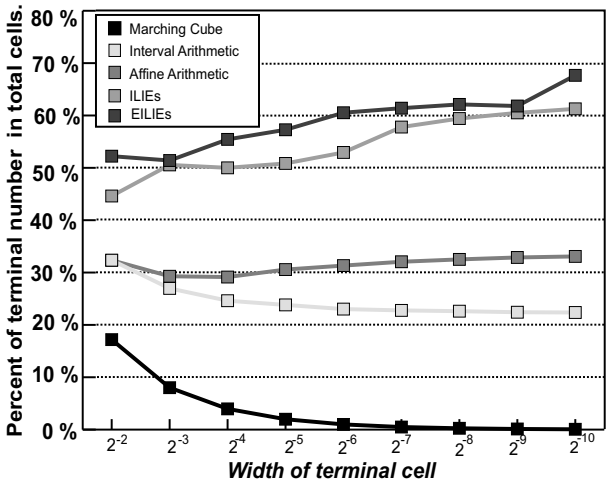

Figure 6.13: Percentage of straddling cells in total cell.

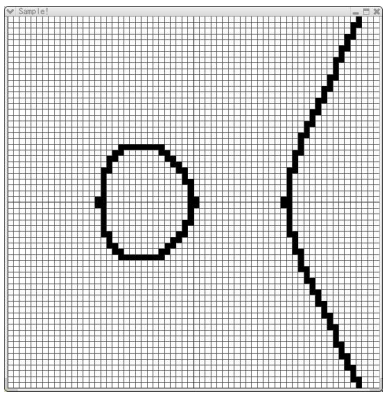

Figure 6.14: Marching Cube.

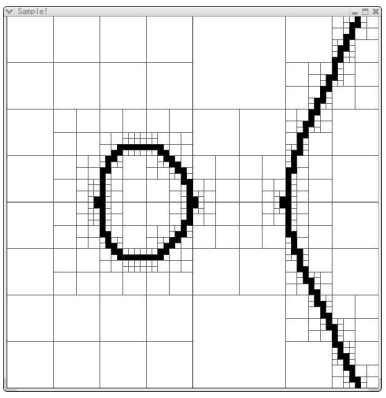

Figure 6.15: Affine Arithmetic.

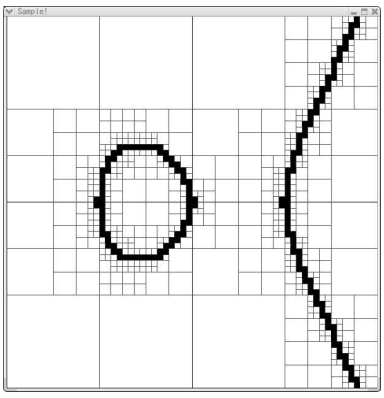

Figure 6.16: Interval Arithmetic.

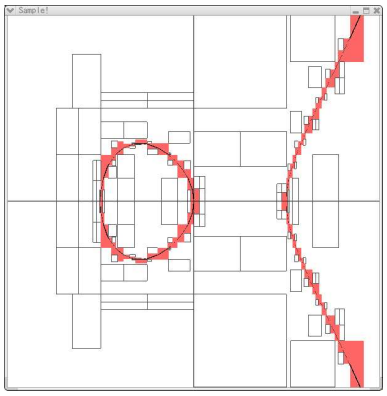

Figure 6.17: ILIE.

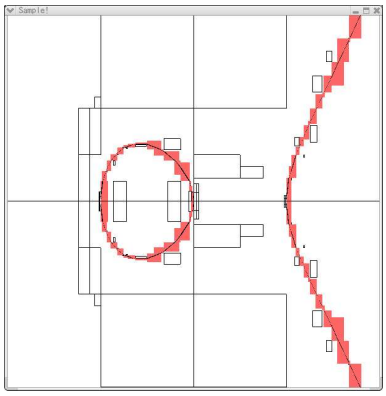

Figure 6.18: EILIE.

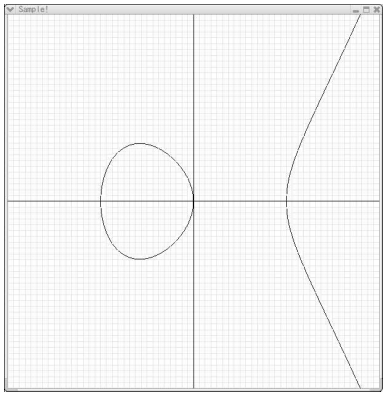

Figure 6.19: Marching Cuble rline.

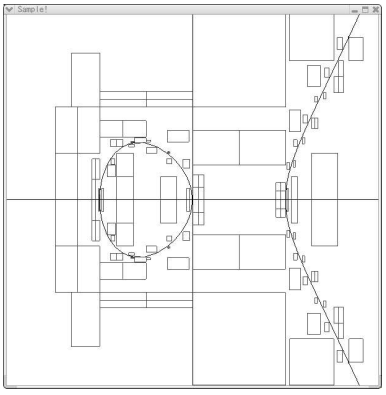

Figure 6.20: ILIE line.

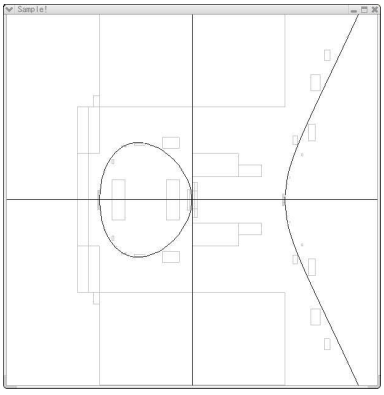

Figure 6.21: EILIE line.

Two dimensional example [53]: $20160x^5 - 30176x^4 + 14156x^3 - 2344x2 + 151x + 237 - 480y$.

The bottom chart shows discrete values with computation times, and the following pages shows image results with each method. In this function, we show 2^{-6}.

Time table (Second)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-2} (0.25)	0.000213	0.001204	0.006588	0.007951	0.011093
2^{-3} (0.125)	0.000726	0.003922	0.016358	0.019389	0.023439
2^{-4} (0.0625)	0.002771	0.012406	0.031775	0.034808	0.039166
2^{-5} (0.03125)	0.011062	0.035600	0.049527	0.054840	0.059306
2^{-6} (0.015625)	0.046004	0.100149	0.075722	0.073517	0.079831
2^{-7} (0.0078125)	0.175570	0.236630	0.119056	0.109680	0.105607
2^{-8} (0.00390625)	0.697627	0.487856	0.191662	0.130660	0.136262
2^{-9} (0.001953125)	18.505796	1.011880	0.333998	0.175303	0.181378
2^{-10} (0.0009765625)		2.021320	0.590873	0.231182	0.225799

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-2}	24 / 256 (2^8)	24 / 160	24 / 232	109 / 160	76 / 139
2^{-3}	46 / 1024 (2^{10})	46 / 547	46 / 583	177 / 406	102 / 289
2^{-4}	90 / 4096 (2^{12})	90 / 1747	90 / 1135	193 / 757	120 / 487
2^{-5}	180 / 16384 (2^{14})	180 / 5077	180 / 1759	264 / 1141	168 / 718
2^{-6}	361 / 65536 (2^{16})	361 / 13954	361 / 2644	375 / 1594	252 / 982
2^{-7}	717 / 262144 (2^{18})	717 / 33595	717 / 4054	602 / 2194	367 / 1285
2^{-8}	1429 / 1048576 (2^{20})	1429 / 71716	1429 / 6538	879 / 2884	552 / 1693
2^{-9}	2852 / 4194304 (2^{22})	2852 / 147400	2852 / 11161	1312 / 3802	864 / 2242
2^{-10}	5699 / 16777216 (2^{24})	5699 / 298507	5699 / 20041	2033 / 5122	1179 / 2797

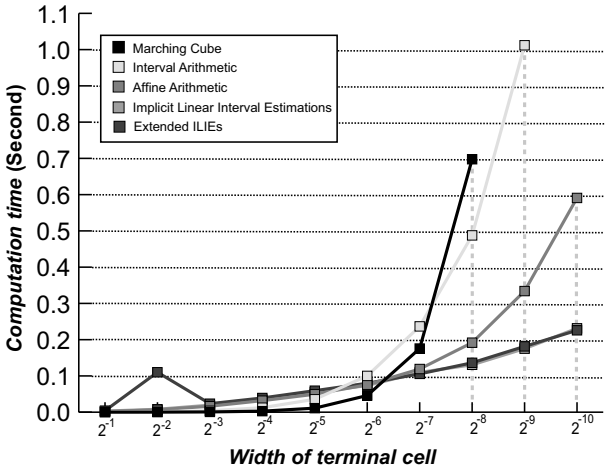

Figure 6.22: Calculation time with each polygonization result.

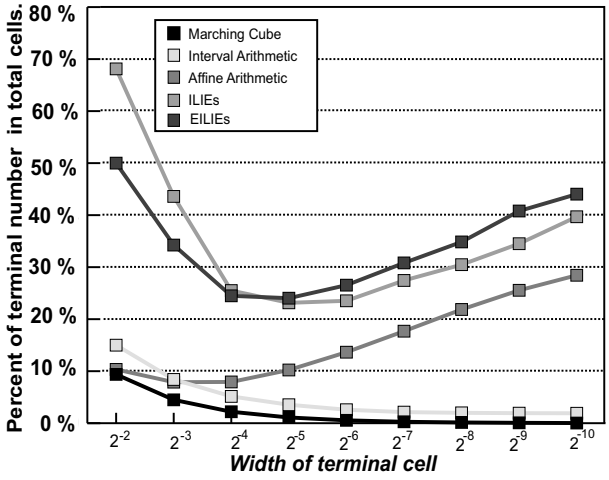

Figure 6.23: Percentage of straddling cells in total cell.

Figure 6.24: Marching Cube.

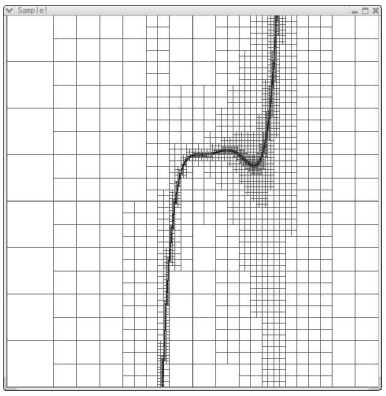

Figure 6.25: Affine Arithmetic.

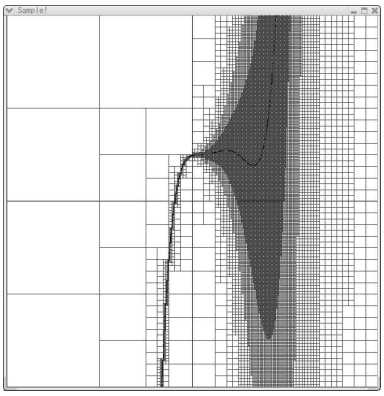

Figure 6.26: Interval Arithmetic.

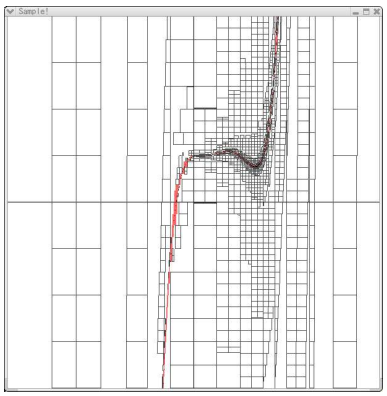

Figure 6.27: ILIE.

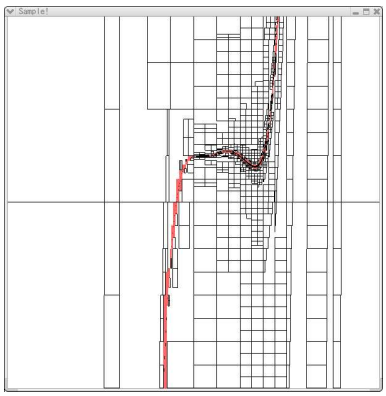

Figure 6.28: EILIE.

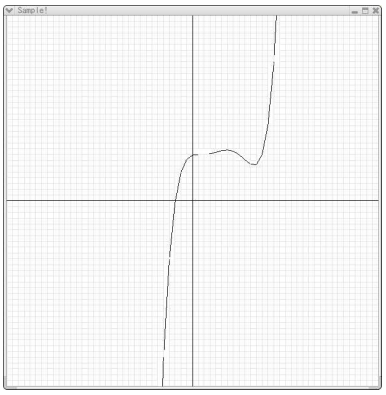

Figure 6.29: Marching Cuble rline.

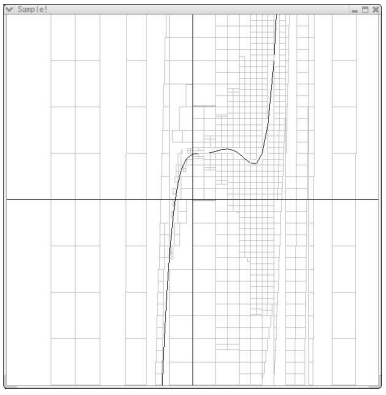

Figure 6.30: ILIE line.

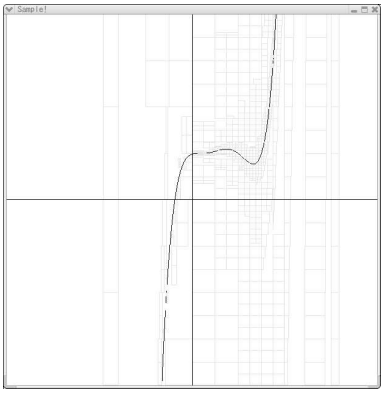

Figure 6.31: EILIE line.

This is a simple two dimensional exmples, which is related to our suggested problem in our Introduction. The figures below are described by calculating $y = x$. We show the Interval Arithmetic cell division result and ILIE polygonization result, and four methods for the computation time with a line graph. The ILIE result is made with a continuous line, not constructed by line segments.

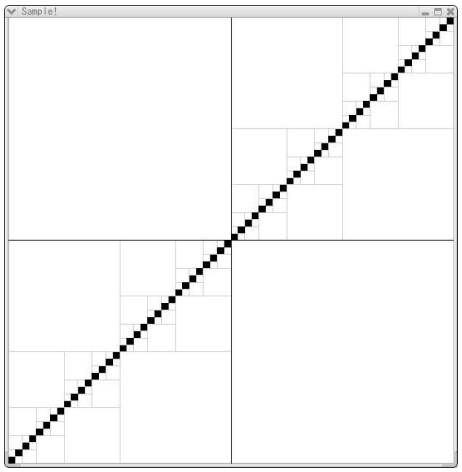

Figure 6.32: Interval Arithmetic.

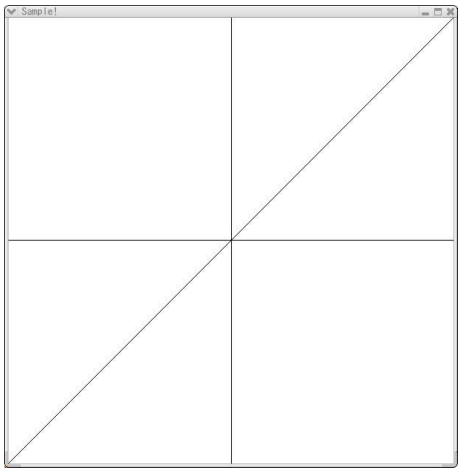

Figure 6.33: ILIE result.

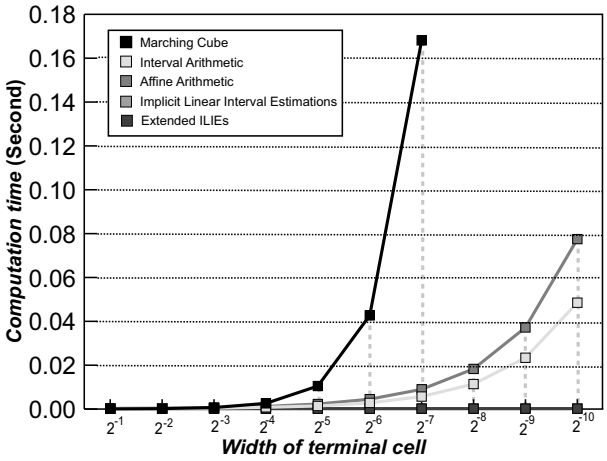

Figure 6.34: Calculation time with each polygonization result.

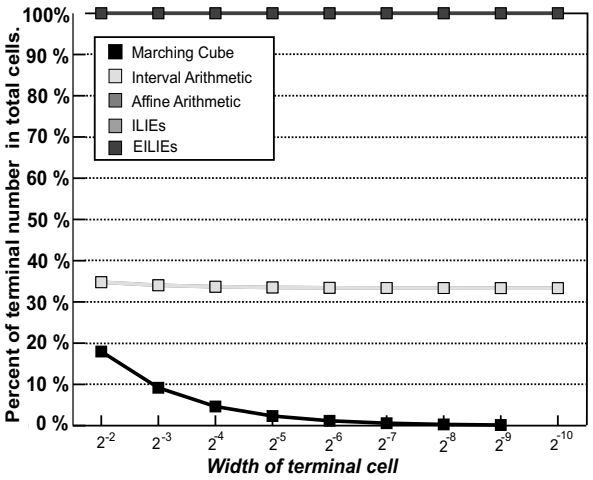

Figure 6.35: Percentage of straddling cells in total cell.

The calculation times of ILIE with this formula are stable, because the decomposition step is stopped by curvature analysis in the first step.

Bicorn formula [52]: $y^2(0.75^2 - x^2) - (x^2 + 2.0 \times 0.75y - 0.75^2)^2$.

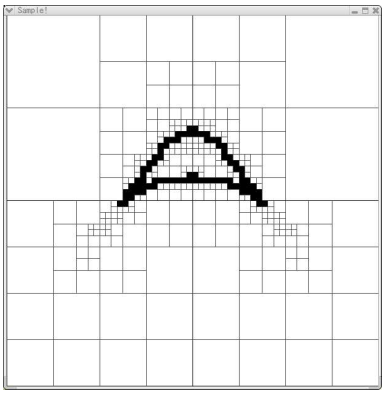

Figure 6.36: IA.

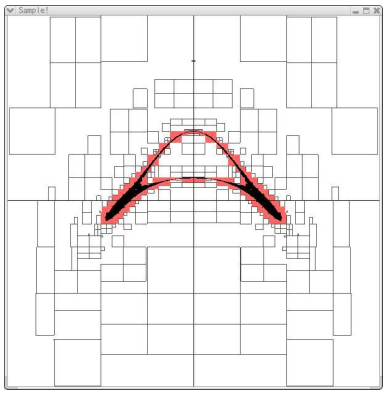

Figure 6.37: ILIE.

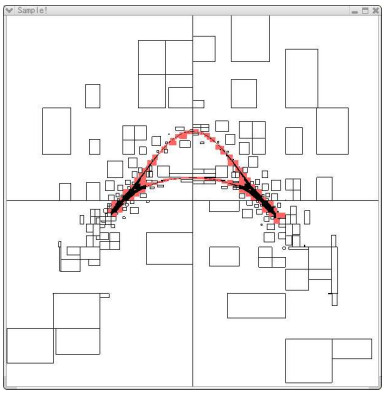

Figure 6.38: ELIEs.

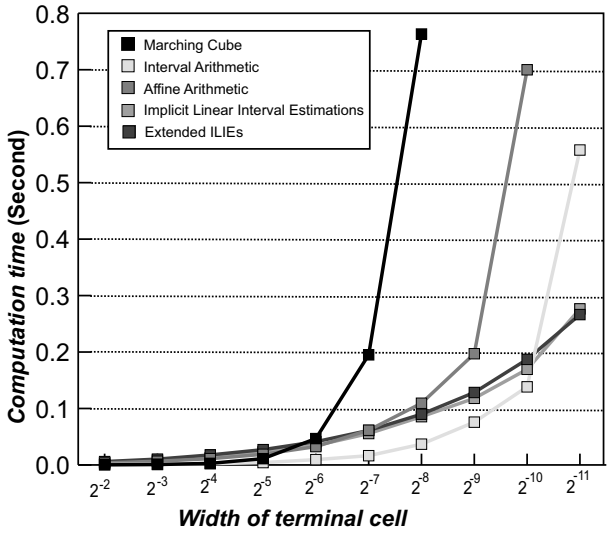

Figure 6.39: Calculation time with each polygonization result.

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-4}	82 / 4096 (2^{12})	80 / 376	82 / 541	136 / 382	119 / 325
2^{-5}	160 / 16384 (2^{14})	158 / 718	160 / 925	240 / 628	198 / 505
2^{-6}	318 / 65536 (2^{16})	316 / 1414	318 / 1573	430 / 1018	319 / 760
2^{-7}	636 / 262144 (2^{18})	634 / 2800	636 / 2794	664 / 1498	494 / 1126
2^{-8}	1272 / 1048576 (2^{20})	1270 / 5590	1272 / 4951	1094 / 2322	756 / 1630
2^{-9}	2548 / 4194304 (2^{22})	2546 / 11188	2548 / 9157	1590 / 3526	1126 / 2314
2^{-10}	5104 / 16777216 (2^{24})	5102 / 22348	5104 / 17305	2372 / 4684	1670 / 3307

"Clown smile" formula [52]: $y^2(0.75^2 - x^2) - (x^2 + 2.0 \times 0.75y - 0.75^2)^2$.

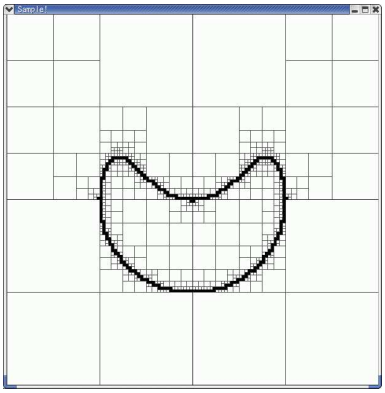

Figure 6.40: IA.

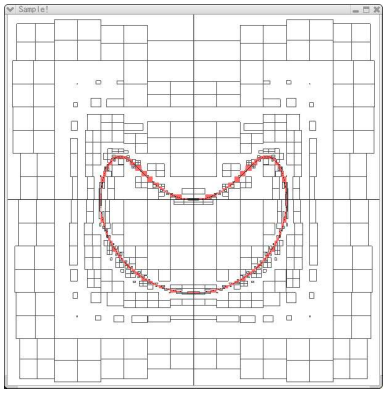

Figure 6.41: ILIE.

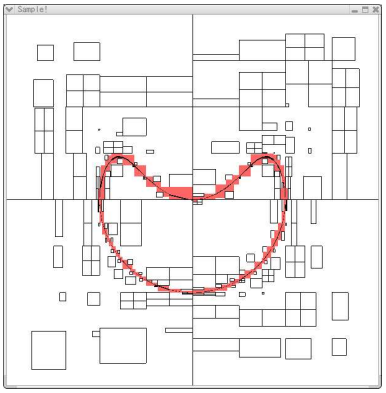

Figure 6.42: ELIEs.

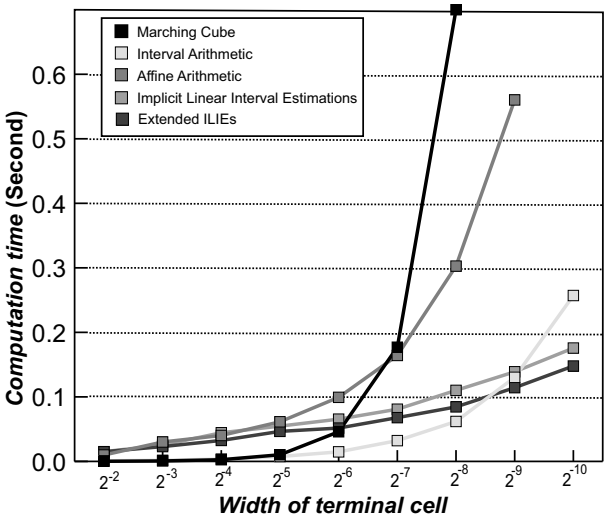

Figure 6.43: Calculation time with each polygonization result.

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-4}	130 / 4096 (2^{12})	126 / 400	130 / 772	158 / 622	138 / 358
2^{-5}	254 / 16384 (2^{14})	250 / 820	254 / 1468	316 / 910	218 / 481
2^{-6}	506 / 65536 (2^{16})	502 / 1630	506 / 2350	442 / 1144	269 / 556
2^{-7}	1010 / 262144 (2^{18})	1010 / 3982	717 / 4054	593 / 1378	393 / 745
2^{-8}	2018 / 1048576 (2^{20})	2012 / 6514	2018 / 7138	928/ 1918	511 / 928
2^{-9}	4030 / 4194304 (2^{22})	4024 / 13060	2852 / 11161	1184 / 2314	709 / 1225
2^{-10}		8053 / 25063	5699 / 20041	1630 / 3022	967 / 1606

"Cubic" formula [52]: $f(x, y) = y^2 - x^3 + x - 0.5$.

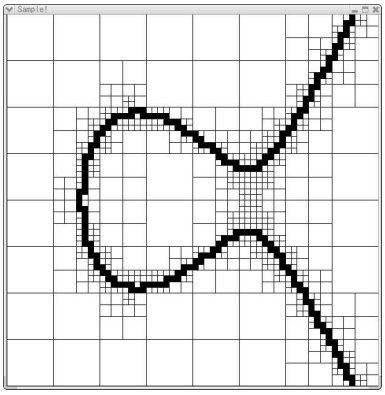

Figure 6.44: IA.

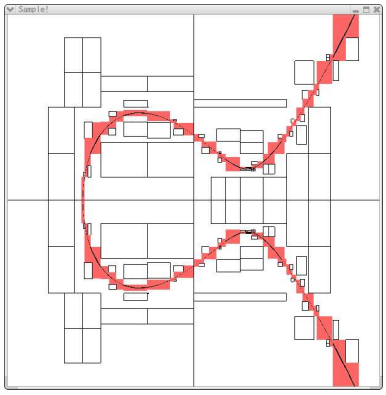

Figure 6.45: ILIE.

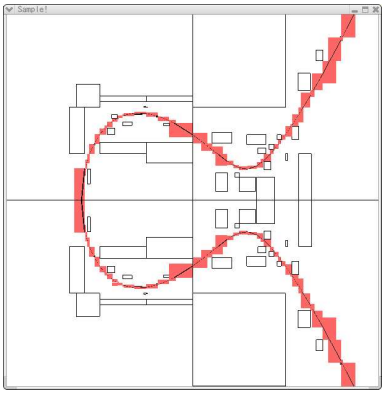

Figure 6.46: ELIEs.

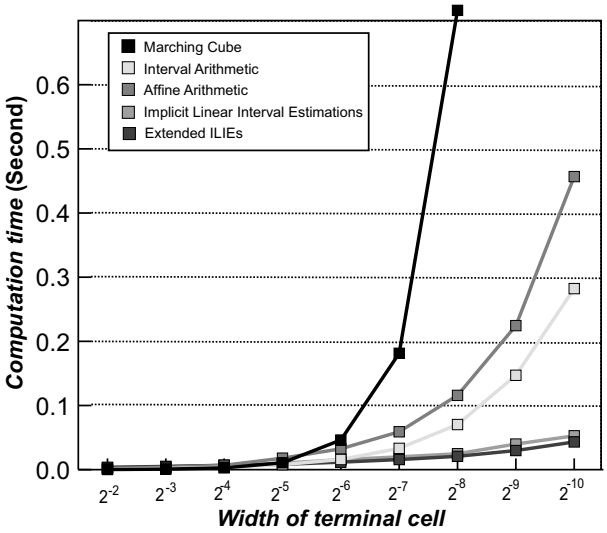

Figure 6.47: Calculation time with each polygonization result.

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-4}	198 / 4096 (2^{12})	198 / 820	198 / 616	178 / 298	105 / 172
2^{-5}	396 / 16384 (2^{14})	396 / 1708	396 /1234	258 / 442	129 / 208
2^{-6}	788 / 65536 (2^{16})	788 / 3526	788 / 2446	332 / 562	189 / 298
2^{-7}	1570 / 262144 (2^{18})	1570 / 7090	1570 / 4828	494 / 808	241 / 376
2^{-8}	3140 / 1048576 (2^{20})	3140 / 14158	3140 / 9590	640 / 1036	346 / 538
2^{-9}	6280 / 4194304 (2^{22})	6280 / 28618	6280 / 18970	962 / 1510	498 / 760
2^{-10}		12558 /57466	12558 / 37810	1402 / 2188	719 / 1090

"Pear" formula [52]: $f(x, y) = 4y^4 - (x+1)^3(1-x)$.

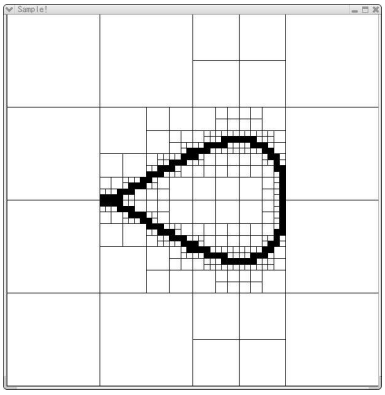

Figure 6.48: IA.

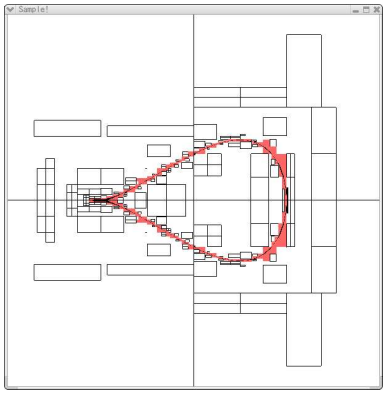

Figure 6.49: ILIE.

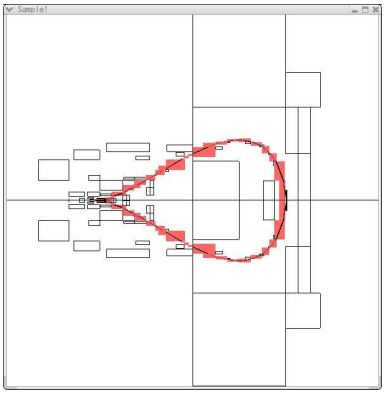

Figure 6.50: ELIEs.

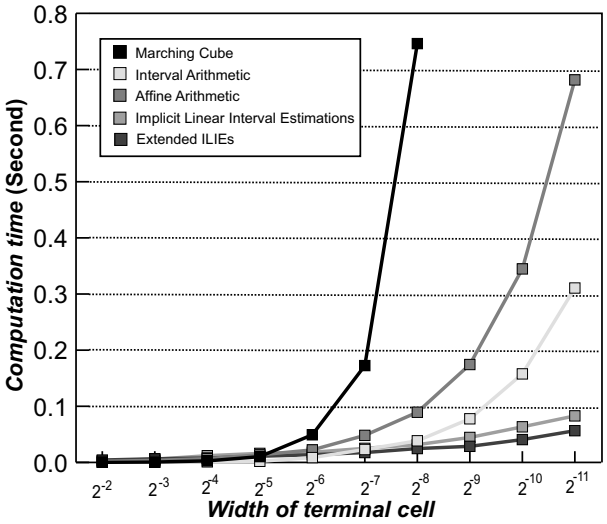

Figure 6.51: Calculation time with each polygonization result.

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-4}	90 / 4096 (2^{12})	90 / 1747	90 / 1135	170 / 370	74 / 172
2^{-5}	180 / 16384 (2^{14})	180 / 5077	180 / 1759	214 / 466	112 / 238
2^{-6}	361 / 65536 (2^{16})	361 / 13954	361 / 2644	306 / 628	142 / 292
2^{-7}	717 / 262144 (2^{18})	717 / 33595	717 / 4054	466 / 898	196 / 376
2^{-8}	1429 / 1048576 (2^{20})	1429 / 71716	1429 / 6538	614 / 1156	288 / 520
2^{-9}	2852 / 4194304 (2^{22})	2852 / 147400	2852 / 11161	832 / 1522	347 / 610
2^{-10}	5699 / 16777216 (2^{24})	5699 / 298507	5699 / 20041	1202 / 2086	507 / 856

6.1.1 Three dimensional polygonization result

In this section, some polygonization experimental results are shown with Marching Cube(MC), Interval Arithmetic(IA), Affine Arithmetic(AA), Implicit Linear Interval Estimations(ILIEs) and Extended Implicit Linear Interval Estimations(EILIEs). All image samples are made with the 0.625 fineness, and ILIEs and EILIEs criteteria in adaptive calculation is fifth of required fineness, $0.625/5.0$.
First three examples are tipical cases in, discrete surface, effective result with ILIE and EILIEs, and ineffective result. Each three case, the function is defined in the top in the left, and detialed time and cell number in total and terminal cell in each levels is described in tables. * in the table means that those value is over 60 seconds or more. Two line graph shows the increasing of computation time and total cell number, and three cell and polygons result are shown in right pages.
After three results, four results are discribed with three cell and polygonization results and line graph of computation time and total cell size.
Unfortunately ILIEs and EILIEs application is not implemented crack avoiding yet. So, some polygonization result yields some crack.

Surfaces: $f(x, y, z) = x^2 + y^2 + z^2 + xyz - 0.5x^2y^2z^2 - 0.25$

Time table (Second)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-1} (0.5)	0.001708	0.003821	0.011546	0.032837	0.026972
2^{-2} (0.25)	0.009219	0.01860	0.035865	0.092322	0.084442
2^{-3} (0.125)	0.069573	0.061045	0.125418	0.252649	0.251939
2^{-4} (0.0625)	0.504225	0.2221697	0.406078	0.876449	0.770895
2^{-5} (0.03125)	*	0.810032	1.458436	2.142245	1.386609
2^{-6} (0.015625)	*	*	5.537661	3.395175	2.637882
2^{-7} (0.0078125)	*	*	*	7.326404	5.089692

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	ELIEs
2^{-1}	112 / 512 (2^9)	112 / 512	112 / 512	112 / 512	112 / 351
2^{-2}	328 / 2388 (2^{12})	328 / 2388	328 / 1632	328 / 1324	368 / 1858
2^{-3}	1300 / 8156 (2^{15})	1300 / 8156	1300 / 4880	1456 / 3816	1494 / 3466
2^{-4}	5200 / 262144 (2^{18})	5200 / 27756	5200 / 15688	6844 / 14148	5784 / 10480
2^{-5}	*	20848 / 103972	20848 / 56400	19384/ 34952	11154 / 19258
2^{-6}	*	*	83704 / 209392	29932/ 54384	22034 / 36947
2^{-7}	*	*	*	72136/ 121948	43412 / 71065

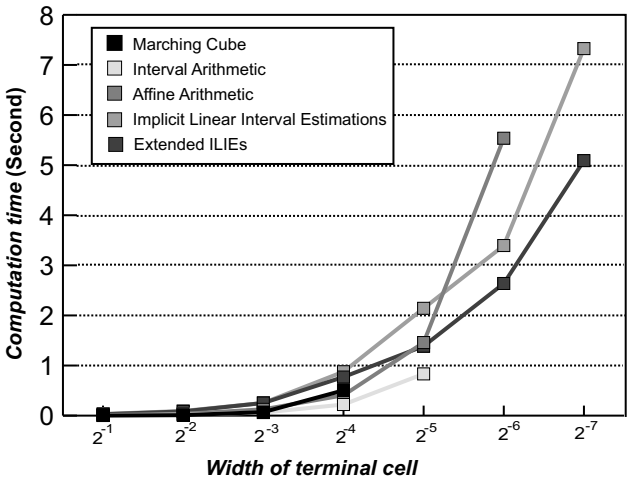

Figure 6.52: Line graph of calculation time with each polygonization result.

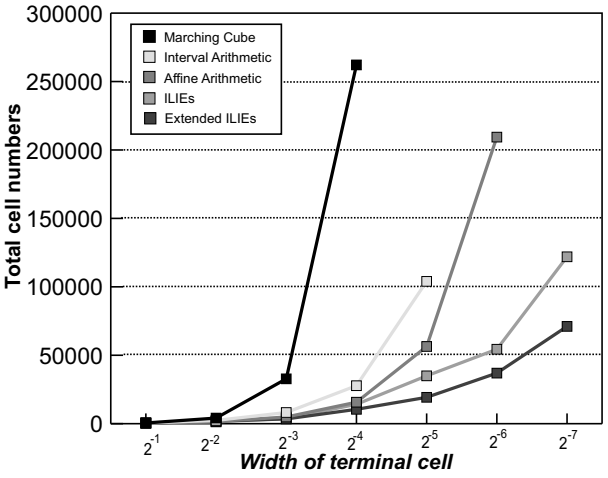

Figure 6.53: Percentage of straddling cells in total cell.

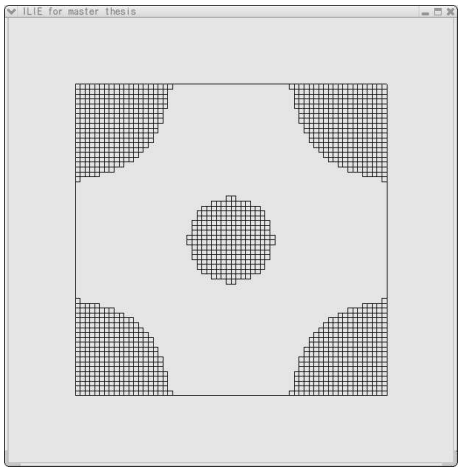

Figure 6.54: MC, IA, AA cell.

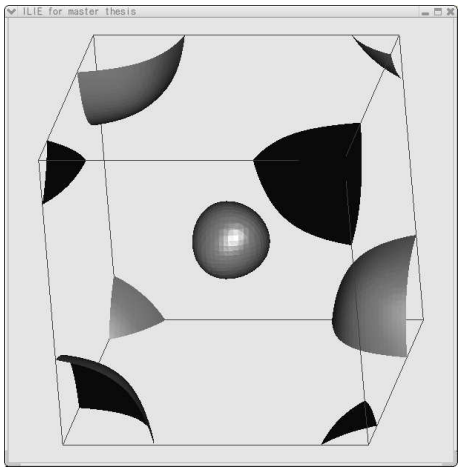

Figure 6.55: MC, IA, AA polygon.

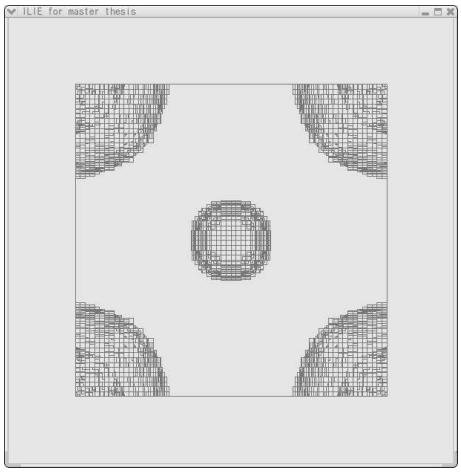

Figure 6.56: ILIE cell.

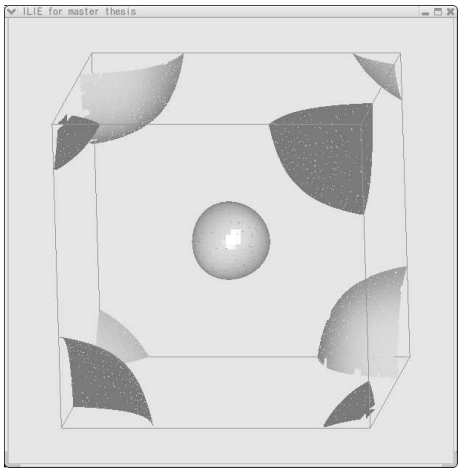

Figure 6.57: ILIE polygon.

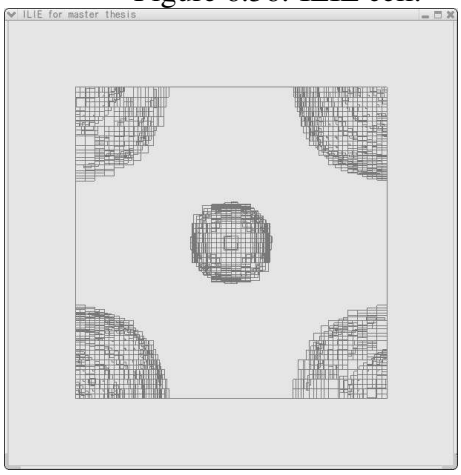

Figure 6.58: EILIE cell.

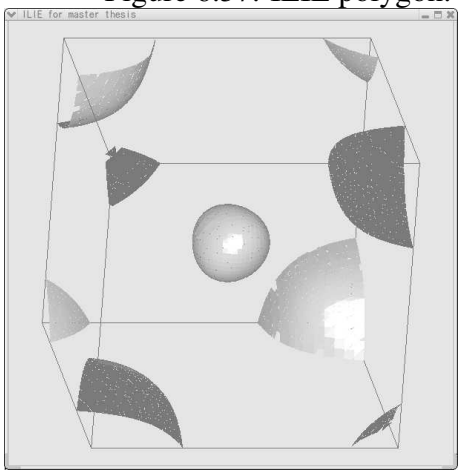

Figure 6.59: EILIE result.

Drop functions: $f(x, y, z) = 4(x^2 + y^2) - (1 + z)(1 - z)^3$;

Time table (Second)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-1} (0.5)	0.001264	0.001456	0.007228	0.016499	0.034700
2^{-2} (0.25)	0.007892	0.004382	0.020472	0.037061	0.037588
2^{-3} (0.125)	0.063833	0.015316	0.052627	0.094670	0.098741
2^{-4} (0.0625)	0.483495	0.061301	0.160347	0.307814	0.132160
2^{-5} (0.03125)	*	0.240768	0.558066	0.459210	0.340750
2^{-6} (0.015625)	*	8.862401	2.175671	1.153318	0.584433
2^{-7} (0.0078125)	*	*	*	1.959882	1.152714
2^{-8} (0.00390625)	*	*	*	4.167993	2.304613
2^{-9} (0.001953125)	*	*	*	8.216114	5.988484

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	ELIEs
2^{-1}	48 / 512 (2^9)	48 / 260	48 / 428	48 / 288	52 / 176
2^{-2}	152 / 4096 (2^{12})	152 / 708	152 / 1128	152 / 708	212 / 540
2^{-3}	592 / 32768 (2^{15})	592 / 2276	592 / 2692	708 / 1912	712 / 1492
2^{-4}	2296 / 262144 (2^{18})	2296 / 7960	2296 / 7624	2772 / 5524	1084 / 2164
2^{-5}	*	9112 / 30248	9112 / 25376	4292 / 8436	2888 / 4908
2^{-6}	*	36544 / 120632	36544 / 91848	11896/ 20980	4924 / 8408
2^{-7}	*	*	*	20516/ 35568	10572 / 17256
2^{-8}	*	*	*	44604/ 74320	22144 / 35400
2^{-9}	*	*	*	92884/ 151404	40804 / 64716

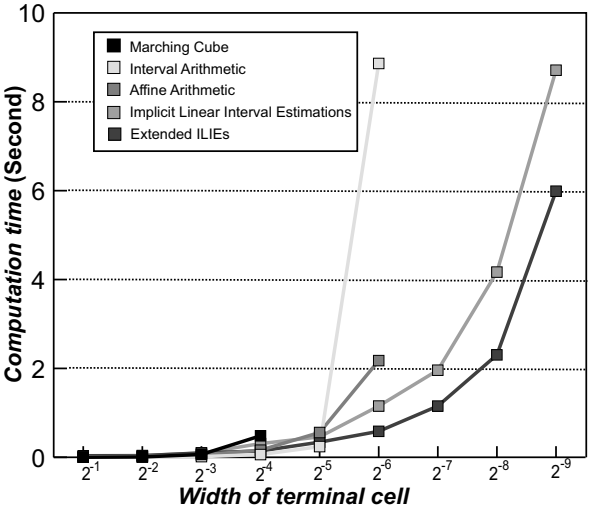

Figure 6.60: Line graph of calculation time with each polygonization result.

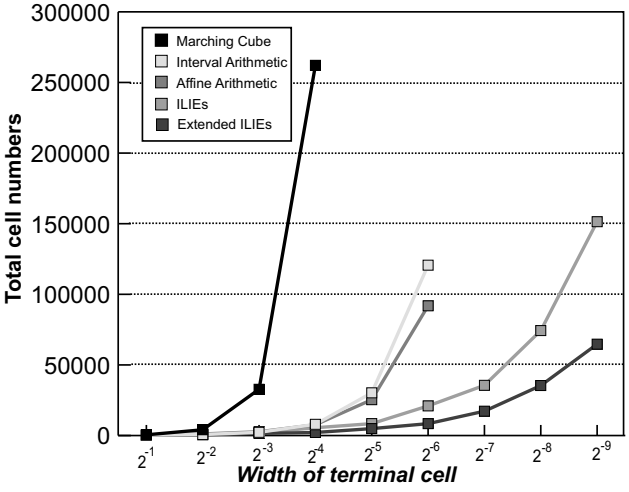

Figure 6.61: Percentage of straddling cells in total cell.

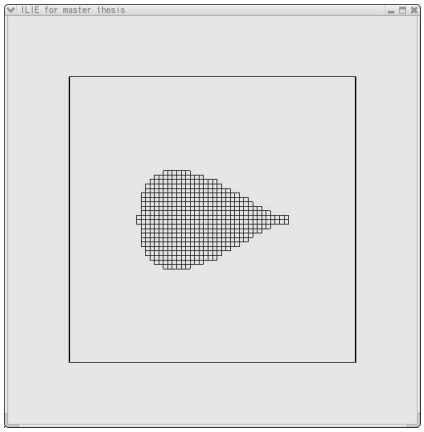

Figure 6.62: MC, IA, AA cell.

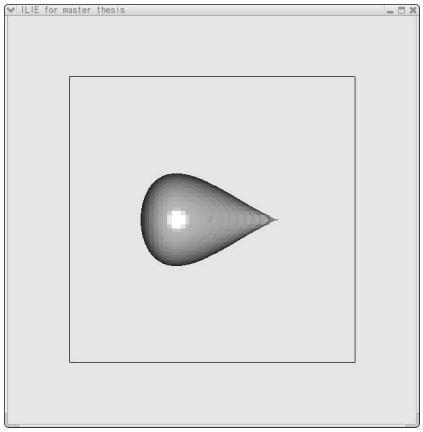

Figure 6.63: MC, IA, AA polygon result.

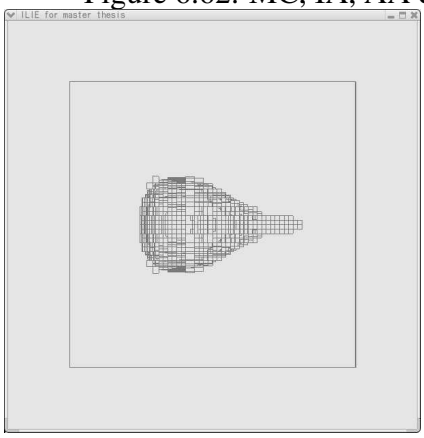

Figure 6.64: ILIE cell.

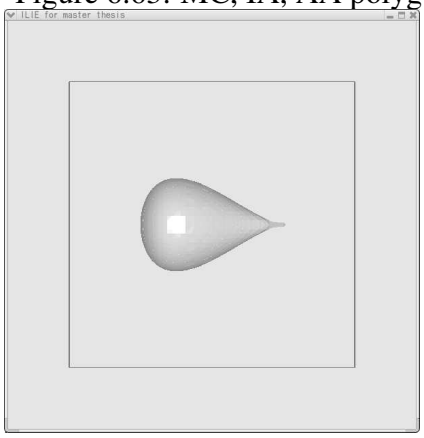

Figure 6.65: ILIE Polygons.

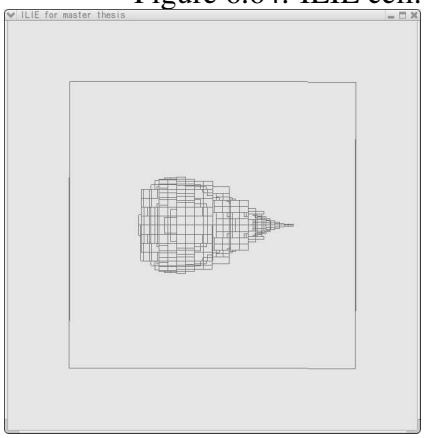

Figure 6.66: EILIE cell.

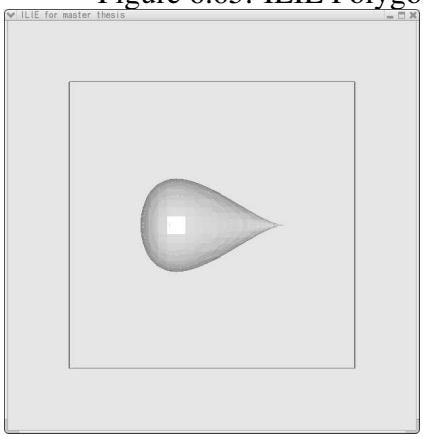

Figure 6.67: EILIE Polygons.

Double tourus functions: $f(x, y, z) = (4x^2(1 - x^2) - y^2)^2 + z^2 - 0.25$

Time table (Second)

Width	Marching Cube	IA	AA	ILIEs	EILIEs
2^{-1} (0.5)	0.001579	0.002736	0.013534	0.052424	0.049948
2^{-2} (0.25)	0.009559	0.014891	0.102399	0.267190	0.176058
2^{-3} (0.125)	0.067686	0.053017	0.482193	0.709181	0.448062
2^{-4} (0.0625)	0.506720	0.194284	1.125623	1.655844	1.221348
2^{-5} (0.03125)	3.806242	0.771984	2.851743	4.752474	2.997970
2^{-6} (0.015625)	*	3.007264	8.61854	12.249875	5.942136

Cell Number (straddling cell / total cell)

Width	Marching Cube	IA	AA	ILIEs	ELIEs
2^{-1}	88 / 512 (2^9)	88 / 344	88 / 512	88 / 512	88 / 512
2^{-2}	328 / 4096 (2^{12})	328 / 1744	328 / 4040	328 / 3928	332 / 2206
2^{-3}	1456 / 32768 (2^{15})	1456 / 5944	1456 / 20448	1464 / 10592	1680 / 5776
2^{-4}	6192 / 262144 (2^{18})	6192 / 22128	6192 / 47328	6404 / 25320	6702 / 15856
2^{-5}	25496/ 2097152 (2^{21})	25496 / 86136	25496 / 113184	30632 / 72528	20050 / 40048
2^{-6}	*	103736/ 340992	103736 / 328336	99136 / 192872	42214 / 112886

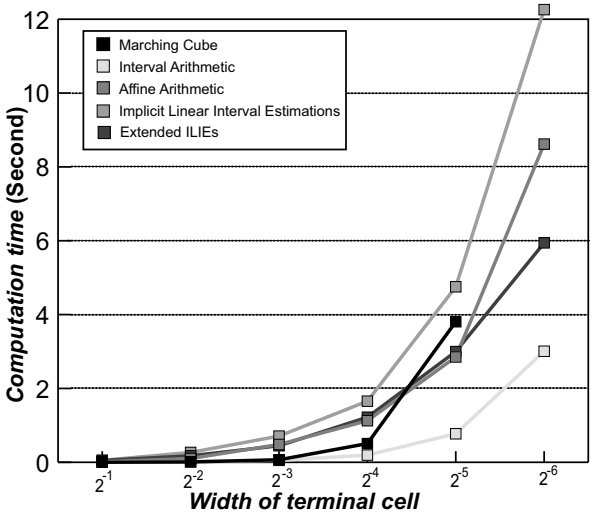

Figure 6.68: Line graph of calculation time with each polygonization result.

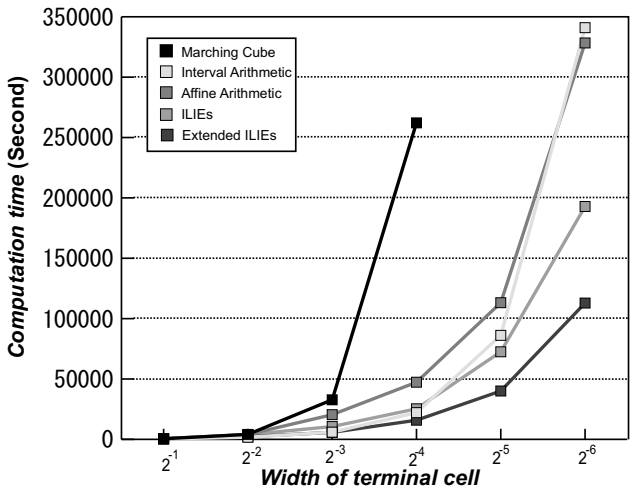

Figure 6.69: Percentage of straddling cells in total cell.

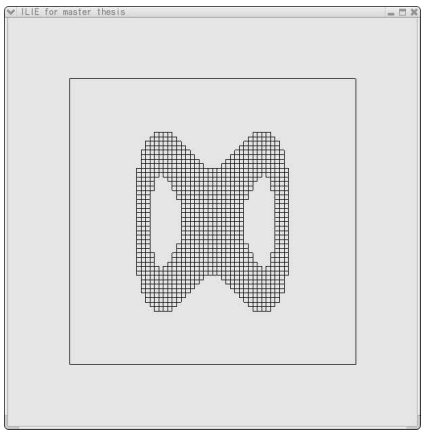

Figure 6.70: MC, AA, IA cell.

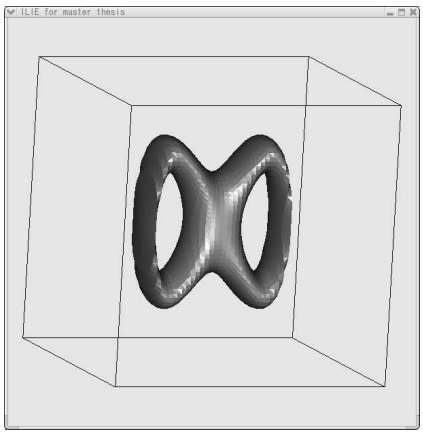

Figure 6.71: MC, AA, IA polygon.

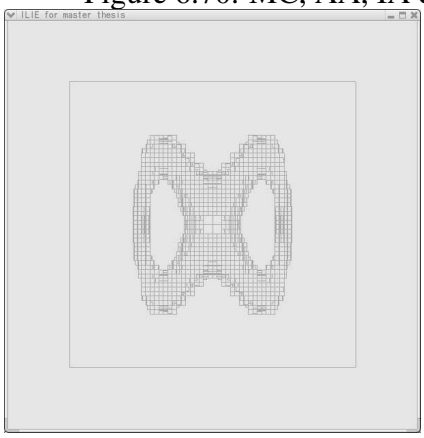

Figure 6.72: ILIE cell.

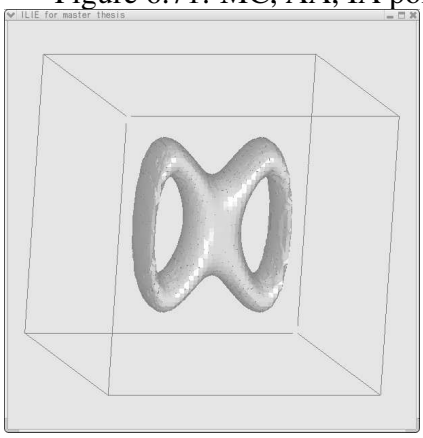

Figure 6.73: ILIE polgyon.

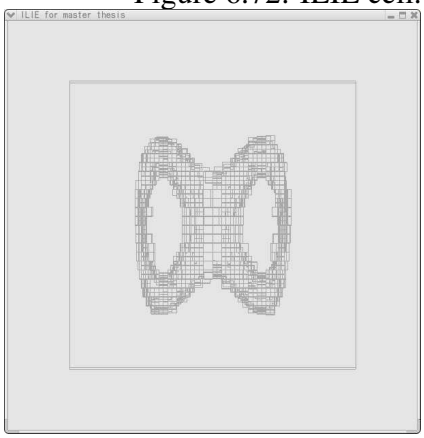

Figure 6.74: EILIE cell.

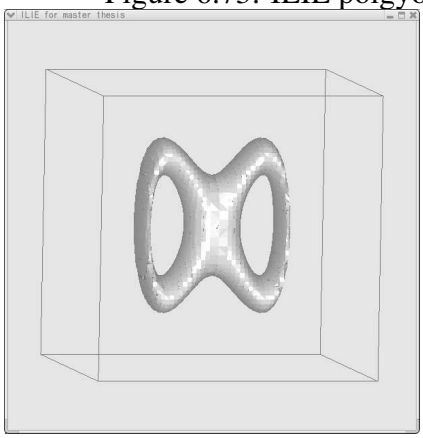

Figure 6.75: EILIE polygon.

Tablet $f(x, y, z) = 1 - z^4 - y^2 - x^2$

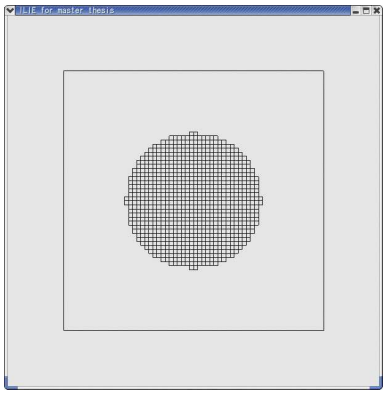

Figure 6.76: MC, IA, AA cell.

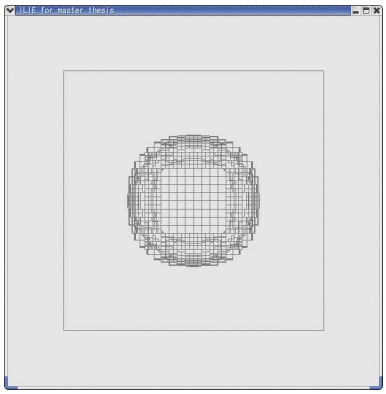

Figure 6.77: ILIEs cell.

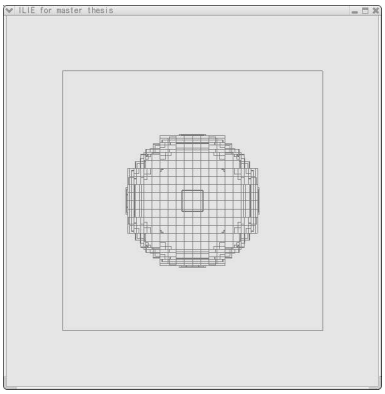

Figure 6.78: ELIEs cell.

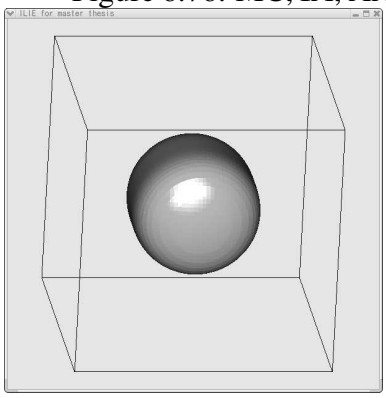

Figure 6.79: MC, IA, AA.

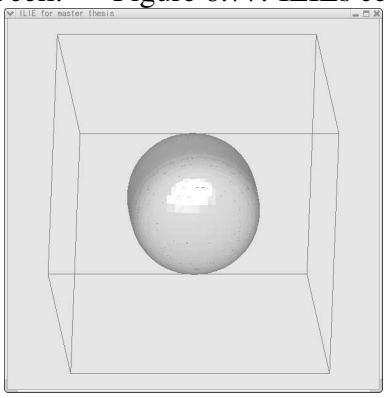

Figure 6.80: ILIEs polgyon.

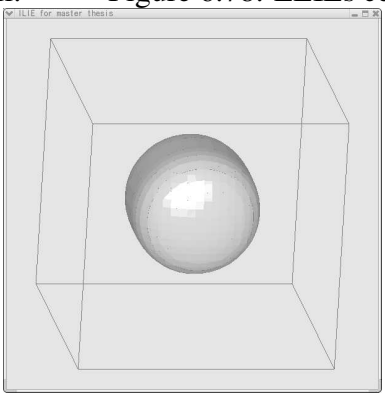

Figure 6.81: ELIEs polygon.

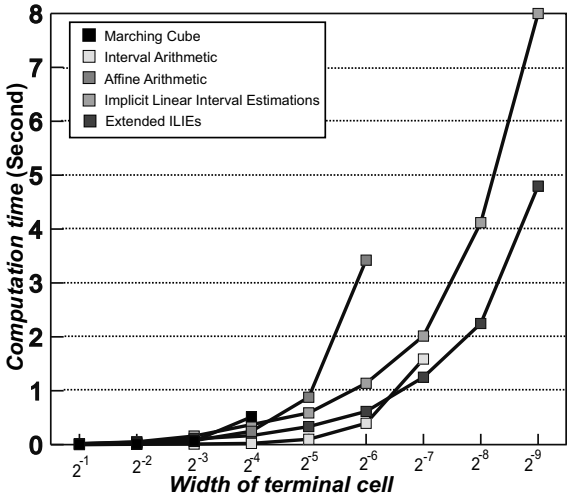

Figure 6.82: Calculation time.

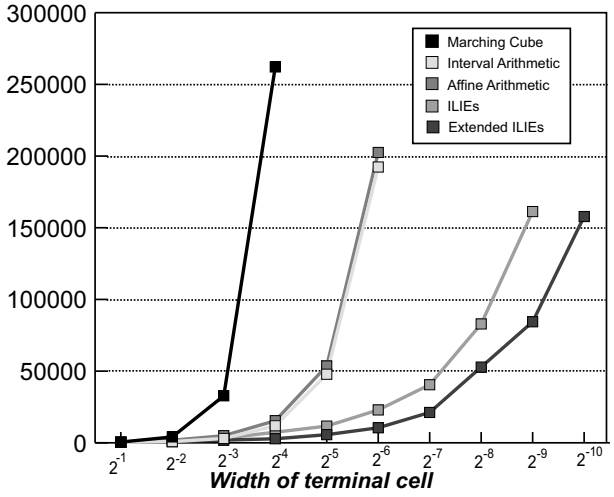

Figure 6.83: Total cell number.

mount & ball $f(x, y, z) = z^2 + y^2 - x^3 + x$

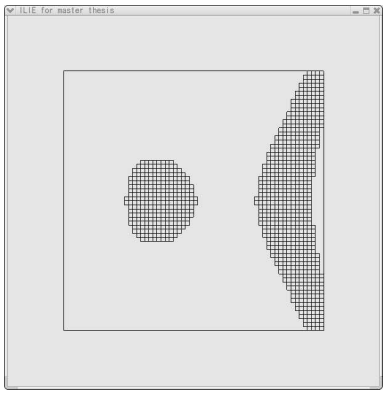

Figure 6.84: MC, IA, AA cell.

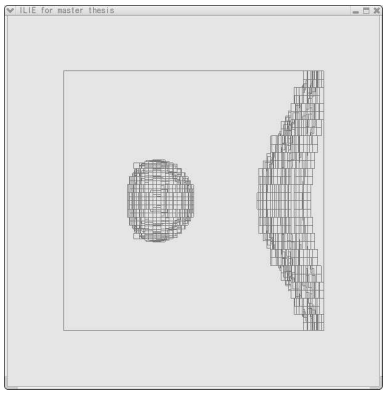

Figure 6.85: ILIEs cell.

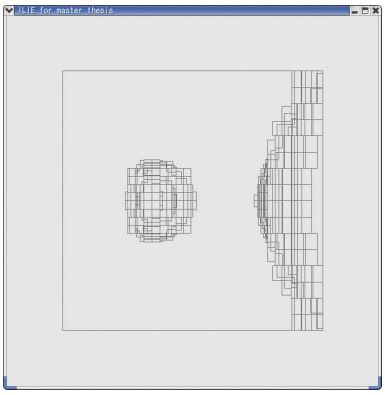

Figure 6.86: ELIEs cell.

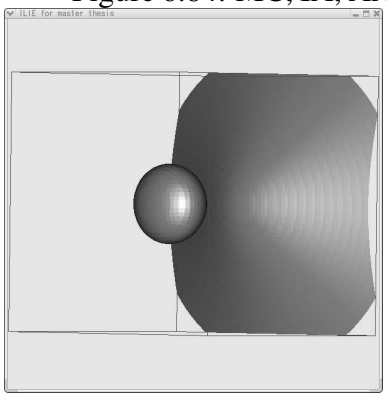

Figure 6.87: MC, IA, AA.

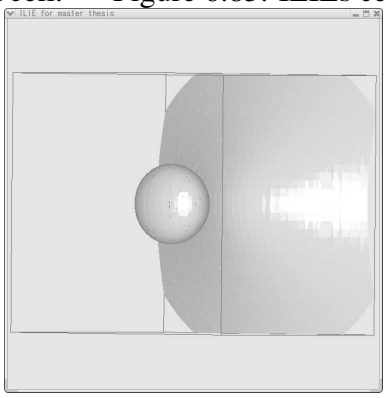

Figure 6.88: ILIEs polygon.

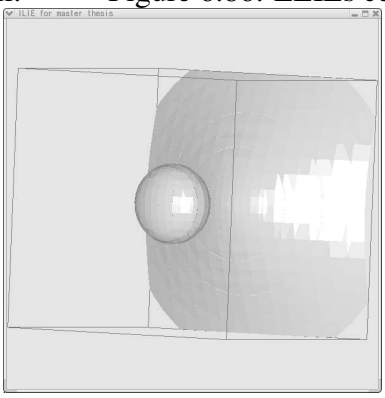

Figure 6.89: ELIEs polygon.

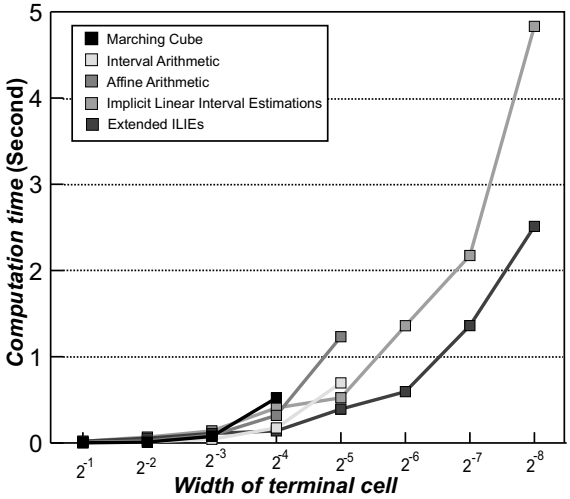

Figure 6.90: Calculation time.

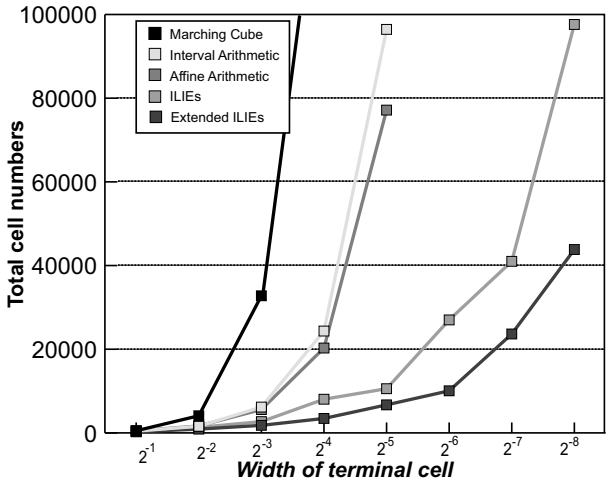

Figure 6.91: Total cell number.

Cross Cap functions: $f(x, y, z) = 4x^2(x^2 + y^2 + z^2 + z) + y^2 * (y^2 + z^2 - 1)$.

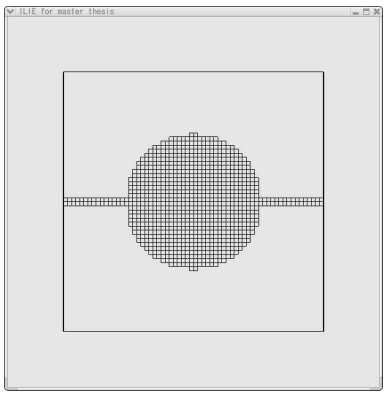

Figure 6.92: MC, IA, AA cell.

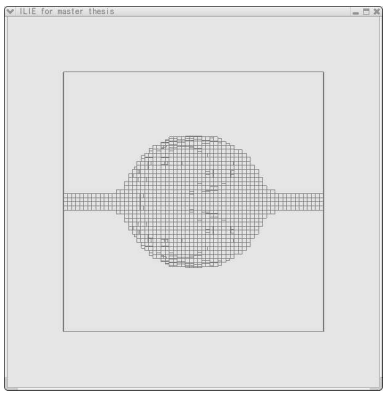

Figure 6.93: ILIEs cell.

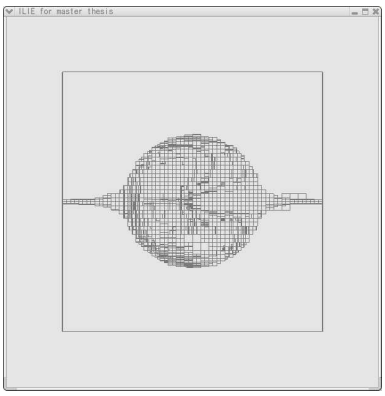

Figure 6.94: ELIEs cell.

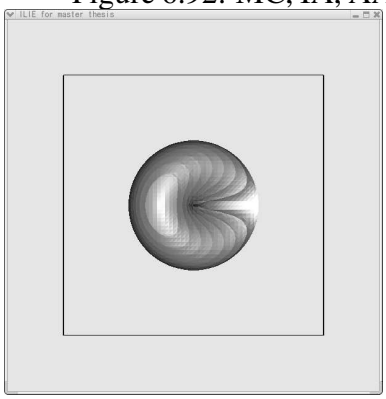

Figure 6.95: MC, IA, AA.

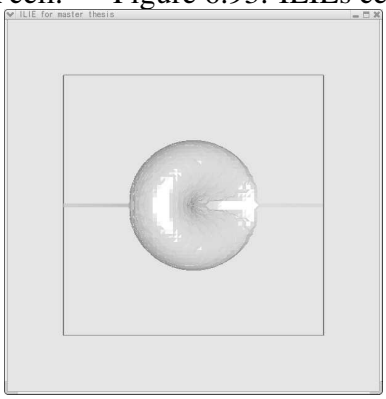

Figure 6.96: ILIEs polygon.

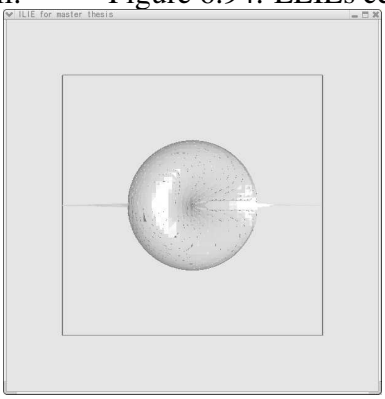

Figure 6.97: ELIEs polygon.

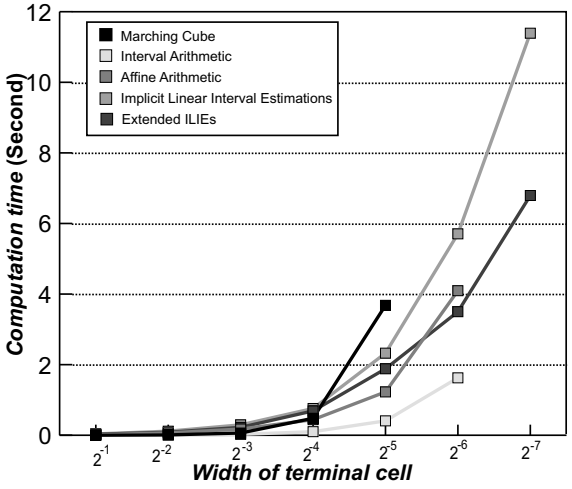

Figure 6.98: Calculation time.

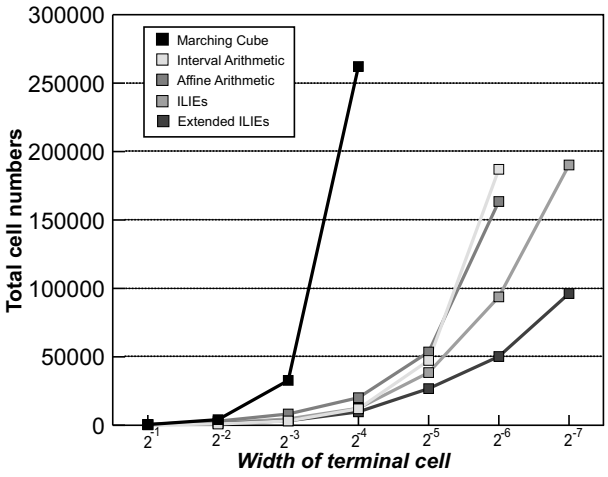

Figure 6.99: Total cell number.

Cylinder formula: $f(x, y, z) = 1.0^2 - x^2 - y^2$

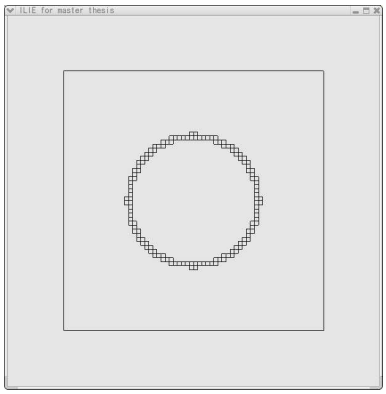

Figure 6.100: MC, IA, AA.

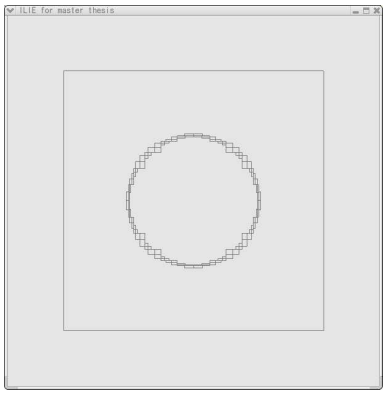

Figure 6.101: ILIEs cell.

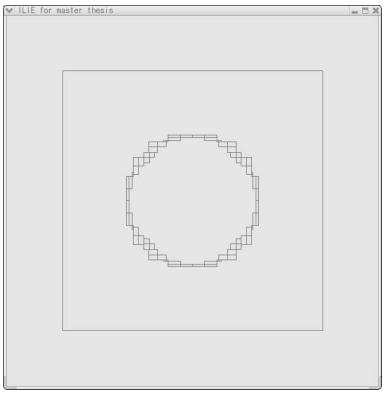

Figure 6.102: ELIEs cell.

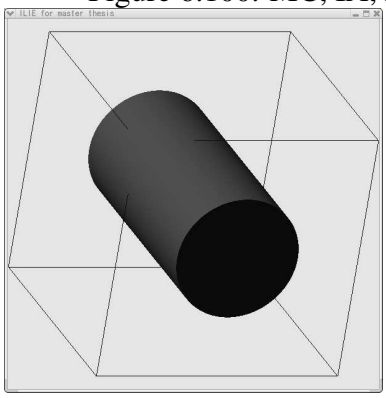

Figure 6.103: MC, IA, AA.

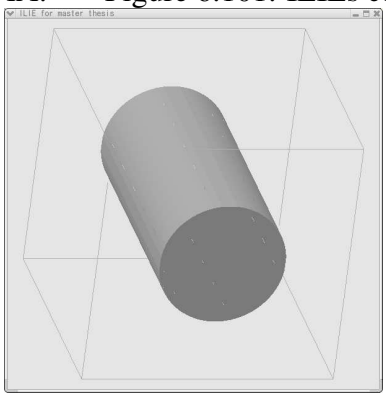

Figure 6.104: ILIEs polygon.

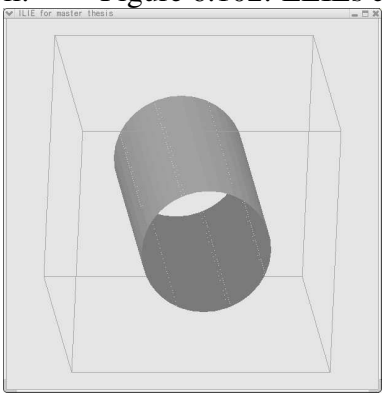

Figure 6.105: ELIEs polygon.

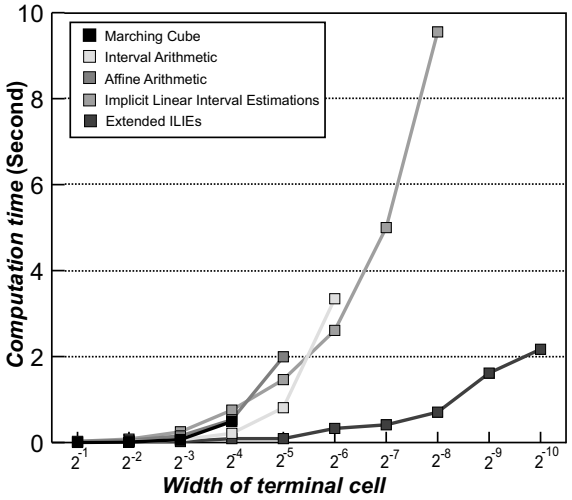

Figure 6.106: Calculation time.

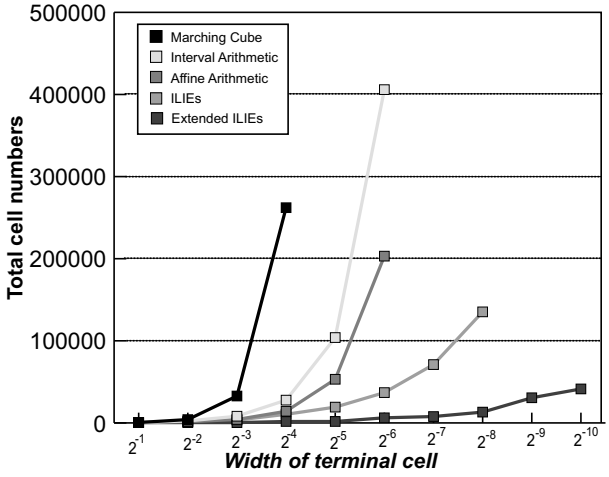

Figure 6.107: Total cell number.

6.2 Discussion

In this section, some discussion with each dimensional result in previous section are described.

6.2.1 Discussion in two dimensional case

For considerating previous section results, we forcused on *computation time*, *cell number* and *required memory* each.

- *Computation time*.
  Marching Cube (MC) method's time is increased about four times by previous situation in all of our test case and this one is the absolutely slowest solution in high accuracy case, but in low case, between 0.25 and 0.0625, this is the most fastest types.
  Next, Affine Arithmetic(AA) and Interval Arihtmetic(IA) results are compared; such comparison has already been done by [52][26].
  In most of our test cases, 7 cases in 8 all cases, Interval Arithmetic is faster than Affine Arithmetic, and Affine Arithmetic is faster than Interval Arithmetic in one complex formulations, Figure 6.21. So, it's so hard to asseverate IA is faster than AA, but it is possible to say that IA is normaly faster than AA. The reason of computation time difference between IA and AA is derived from the basic calculation simplicity in IA described in Chapter 4. IA simply calculate upper and bottom limts in each claculation, and, on the other hand, AA need to more complex calculation related with noise, ϵ. In [52] results, AA's computataion takes five times time of IA. Therefore, in some result's steps, IA's calculation time is half of AA.
  After getting one consideration, IA generally faster than AA, Implicit Linear Interval Estimations(ILIEs) results are considered. Until the certain level, such as 0.015625, ILIEs is the slowest solution compared with AA, IA, and MC. However, ILIEs computation time is not increased as other methods. For all other solutions, time increases geometrically, but ILIEs increasing speed is like arithmetical progression. So, in all of our test cases, ILIEs computation time is stable under 0.5 second, and slowest level zone, under 0.015625, ILIEs computation time in our sample keeps under 0.1 second, and this is not so different from human interaction point of view.
  Extended Implicit Linear Interval Estimations(EILIEs) computation time is not so different from ILIEs time. Basically, EILIEs is a little faster than ILIEs, and in some situation, EILIEs is a little slower than ILIEs. Fortunately, the difference in slower case keeps under 0.01 seconds.

- *Cell number*
  Our results show total cell and terminal cell number. Simply thinking, total cell number is related with the required memory for computing the function, and terminal cell number is related with the result line segments numbers and those file sizes. In detail discussion, MC cell required 28 bytes, IA and AA required 60 bytes, ILIEs and EILIEs required 100 bytes in each cells.

From the viewpoitns of total cells, MC is obviously worst situation, and IA is also worse one. The required memory is increased as geometrical progression. AA is better than IA in memory consumption points.
ILIEs memory consumption is abosultely better than conventional three solutions. ILIEs memory requirement is increased as arithmetic progression.
In EILIEs case, memory consumption is fewer than ILIEs. At an average rate of our result, two third parts of ILIEs memory is consumed for computing with EILIEs. So, EILIEs is the best solution for avoiding extra memory consumption.

Next, terminal cell points is analyzed. In some case, terminal cell has two lines, but it is not frequently. So, we think each terminal cell has one line segment. MC, AA, and IA yields almost same number of segments and result numbers are almost dobuled with decreasing the terminal cell size into half size. On the other hand, ILIEs results is not increased as conventional three solution. In our result, ILIEs semgents number in the minimum level is less than half of conventional solution. In extremely low case, ILIEs terminal cell number is more than conventional solution, but those differecnce is less than one hundred. For getting the stable result numbers, ILIEs is the better solution, and EILIEs yields less number results. In our result average, ELIEs results is two third parts of ILIEs.
Both situation, holding down memory consumption and result segment numbers, EILIEs is the best choice.

- *accuracy*
  As described in Chapter 3, ILIEs and EILIEs yields good segments which including lines. For explaining accuracy points, one example figure is shown in below:

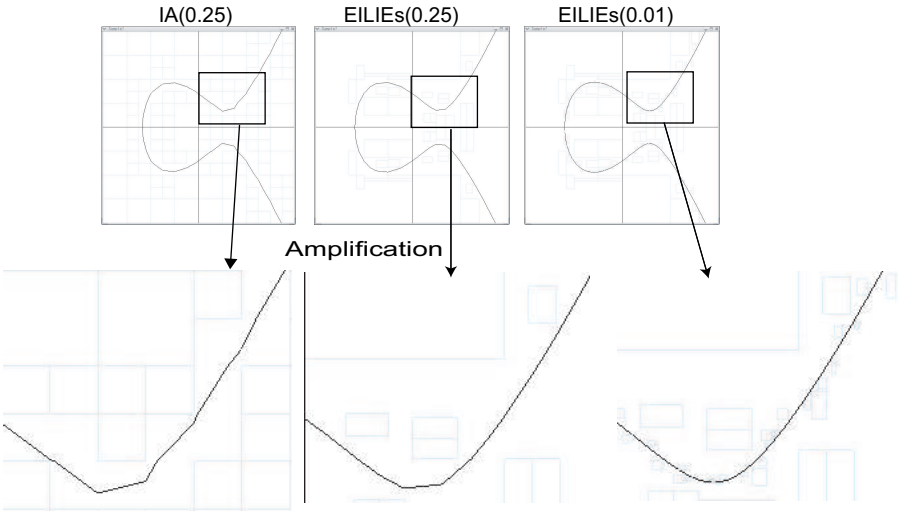

Figure 6.108: The accuracy comparison between IA and EILIEs.

This is the typical case in accuracy comparison. Figure 6.108 shows two results in IA and ELIEs in Cubic formula with 0.25 terminal cell width, and the right image is the fine lines. ELIEs's result is absolutely accuracy than IA. The reason of this result is already mentioned in Chapter 3.2.3, all test cases use linear interpolation for detecting vertex position, and absolutely accurate result is yidled in narrow range case, as ILIEs and EILIEs. So, ILIEs and EILIEs yields more accurate results than conventional three methods in any case.
Next, we discussed about ILIEs and EILIEs with Figure 6.1. From the definition, EILIEs yields tighter range than ILIEs. In the worst case, ELIEs's segment width is equal to ILIEs. First line in Figure 6.1 is the typical case in the narrow range with EILIEs. Therefore, EILIEs is the best choice to get accuracy results in each terminal cell size.

6.2.2 Discussion in three dimensional case

As described in prerequiisite, 6.1.1, our application is not implemented with cracks solution. So, it is difficult to derive some conclusion related with ILIEs and EILIEs. However, it is possilbe to get some deductions. In three dimensional case, we discuss about accuracy, cell number, and computation time. After that we analysis

- *Cell number*.
  In all cases, EILIEs's total cell and terminal cell number ascent is slower than any other solutions. Especially, in complex formulation: Double torus and Cross cap, EILIEs total cell is sixty percent of ILIEs.

- *Computation time*.
  EILIEs is the fastest plygonization technique in most tested cases, excepting Double torus and Cross Cap. So, in simple mathematical formulation, EILIEs is faster than three conventional solutioins, but in complex formulation with many square terms, IA is the fastest solution. The reason for the difference is that IA yields more precise results in square roots, and those value effects total cell numbers. Total cell numbers effects the calculation number, and the important point for getting faster result is to reduce the total numbers. In Double Torus and Cross Cap, there are not much difference between IA and other solution. Besides the cell number, IA is faster than AA, ILIEs, and EILIEs as described in Chapter 4. So, the difference of total cell number is not simply reflected the computation time result.

- *Accuracy*.
  In three dimensional calculation, our application is not implemented for cracks, and this effect some points in polygonization result. So, it is difficult to discuss in polygonization result with comparing conventional solution and our solutions. So, comparions with conventional polygonization are skipped. However, it is possible to discuss ILIEs and EILIEs cell accuracy. In some test case, Drop and Cross Cap, EILIEs cell result is more adaptive for polygonization results, and this effect polygonization accuracy. So, it is possilble to deduce that EILIEs generally derives some better adaptive cell decomposition result than ILIEs.

- *topology*.
  Some topology based polygonization methods are suggested in [42] [43]. This information is useful whether making a hole or connecting surfaces.

From the above analysis, we can conclude that EILIEs has a possibility to be good polygonization, and simple Affine Arihtmetic is not suitable for some polygonization computation.

Chapter 7

Conclusion

7.1 Summary

This thesis proposed two innovative *Adaptive Robust* polygonization algorithms of implicit curve and surface, Implicit Linear Interval Estimations (ILIEs) and Extended Implicit Liner Interval Estimations (EILIEs). Both algorithmes are based on octree decomposition method with Affine Arithmetic calculation. Originally, ILIEs is suggested for ray tracing in implicit surface or lines [53][66], and we implement it into polygonzation technique. EILIEs is a extension of ILIEs; improved points are cell pruning after intersection test with ILIEs and formulation of ILIEs with improving Affine Arithmetic multiplication. ILIEs and EILIEs make it possible for some adaptive solutions; cell pruning, intersection tests, curvature analysis and adaptive vertex calculation. Both methods hold down memory consumption, reduce polygonization result size, and at the same time increase accuracy, and improve performance. However, a new type of crack problem presents itself for which this paper suggest one simple solution, but we do not implement this solution, so our test case can not conclude our case is the best, but we can get one new capability to improve polygonization with implicit surface.

7.2 Conclusion

This paper suggests two methods for making implicit curves and surfaces with Implicit Linear Interval Estimations (ILIEs) and Extended Implicit Linear Interval Estimations (ELIEs).
In implicit curves, our method yields one of the ideal results in accuracy, memory consumption and calculation time points. Our methods, especially ELIEs, are superior to conventional decomposition solution as described Chapter 6.2.1.
In implicit surface, work is on going, because, in using cell pruning, another crack problem appears, and we suggest a simple solution for preventing cracks, but it not implemented. So, our implicit surface polygonization framework tests are not completed yet.
However, our result shows one possibility to save memory consumption and get moderate size polygon compared with conventional solution without any extra computation

which is never mentioned in other papers, and there are some possible to decrease the computation time with implicit surface polygonization.
Finally, we conclude that this paper suggests close to optimal solution in polygonization of implicit curve, and the first step for making ideal polygonization of implicit surface.

7.3 Future Work

This thesis has suggested new polygonization framework for implicit curves and surfaces, and our method has some possibility to be better polygonization. So, some future works listed in below are needed to improve it.

- Crack prevention
  As described in Chapter 3 and conclusion, our implementation is not finished to preventing crack. We suggest the simplest solution for preventing crack in Chapter 3.2.3, but we don't conclude this solution is most suitable for our solution. So, we need more discussion for crack avoiding solution with some references [32][45][49][61][61][56][78].

- Translator
  In our research, we prepared the Affine Arithmetic method manually, but such computation is usually done by a computer program. So, we need to implement a translator. After that, additional measurements should be done with procedural implicit functions.

- polygon making improvement.
  In our test case, exhaustive enumeration method's polygonization[13][57] is used. However, there are one good solution for making some edges and features [61][61][56]. So, we need to implement it in our solution, for deriving more good polygonization surface results.

- Optimization
  Some conventional polygonization methods produce equilateral polygons, resulting in a better appearance. Our polynization method does not have such an attribute. So, for comparing methods in total throughput, we would implement some optimization idea into our method. Especially, we use cell based decomposition, this type of polygonization results definitely have tiny cell. We also would implement one algorithm to transact those cell [60][68][73].

- Some procedurally definition Extension
  We should implement some simple value judgment, inside, outside or on surface, and it is possible extended non-manifold polygonization[25].

- Robust solution extension.
  We use Affine Arithmetic for function calculation, but Affine Arithmetic is not suitable for some region split, such as if statement. It is possible to avoid those problems by computation in each split range. However, we suggest to make another solution for tracking error adapting range dividing.

References

[1] R. E. Wengert *A simple automatic derivative evaluation program*, Communications of the, ACM 7(8), 1964.

[2] R.E.Moore Interval Analysis. Prentice-Hall, 1966. *Interval analysis*, Prentice-Hall, 1966.

[3] Robert H Dargel, Frank R. Loscalzo, Thomas H. Witt, *Automatic Error Bounds on Real Zeros of Rtional Functions*, ACM Volume 9, Number 11, November, 1966.

[4] Allen Reiter, *Programming Interval Arithmetic and Applications*, Proceediings of the 1967 Army NUumerical Analysis Conference.

[5] Yasuo Fujii, Kozo Ichida, Takeshi Kiyono, *A Method for Finding the Greatest Value of a Multivariable Function Using Interval Arithmetic,* IPSJ Vol.18 No.11, 1977

[6] Geoff Wyvill, Craig McPheeters, Brian Wyvill, *Data structure for soft objects,* The Visual Computer, Vol 2, 227-234, 1986

[7] Geoff Wyvill, Craig McPheeters, Brian Wyvill, *Animating soft objects,* The Visual Computer, Vol 2, 234-242, 1986

[8] William E. Lorensen, Harve E. Cline, *MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM,* SIGGRAPH 1987.

[9] Brian Von Herzen, Alan H. Barr *Accurate Triangulations of Deformed, Intersecting Surface,* SIGGRAPH 1987.

[10] Andreas Griewank, *On Automatic Differentiation,* Mathematics and Computer Scienece Division, November 1988.

[11] Jules Bloomenthal, *Polygonization of Implicit Surfaces,* Computer Aided Geometric Design 341-355, 1988.

[12] Werner Rheinboldt *On the computation of multi-dimensional solution manifolds of parameterized equations,* Numerische Mathematik, 53:165-182, 1988.

[13] Pasko A., Pilyugin V., Pokrovskiy V. *Geometric modeling in the analysis of trivariate functions,* Computer & Graphics Vol 12, Nos 3/4 pp.455-465, 1988

[14] Max E. Jerrell, *Function minimization and Automatic Differentiation Using C++,* OOPSLA October 1989,

[15] Devendra Kalra, Alan H. Barr, *Guaranteed Ray Intersections with Implicit Surfaces,* SIGGRAPH 1989,

[16] Luiz Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulos, Luiz Velho *Physically-Based Methods for Polygonization of Implicit Surface* In Graphics Interface, page 250-257, 1992.

[17] Tom Duff, *Interval Arithmetic and Recursive Subdivision for Implicit Function and Constructive Solid Geometry* ACM Computer Graphics 1992 (SIGGRAPH 1992).

[18] John M. Synder, *Interval Analysis For Computer Graphics*, SIGGRAPH 1992.

[19] Joao Luiz Dihl Comba, Jorge Stolfi, *Affine Arithmetic and its Application to Computer Graphics,* Symposium on Computer Graphics and Image Processing 1993, (SIBGRAPI'93).

[20] John C. Hart, *Ray Tracing Implicit Surfaces,* SIGGRAPH 93 Modeling, Visualizing, and Animating Implicit Surfaces course notes

[21] C.W.AM. van Overveld and Brian Wyvill *Shrinkwrap: an adaptive algorithm for polygonizing an implicit surface,* The University of Calgary, Department of computer science, Research Report No. 93/514/19, March 1993.

[22] Jules Bloomenthal *An Implicit Surface Polygonizer,* Graphics GemsIV, New York, Academic Press, 1994.

[23] Andrew P. Witkin and Paul S. Heckbert. *Using particles to sample and control implicit surface*, SIGGRAPH 1994.

[24] Marcus Vinicius Alivim Andrade, Joao Luiz Dihl Comba, Jorge Stolfi, *Affine Arithmetic*, Presented at INTERVAL'94, St. Petersburg (Russia), March 5-10, 1994.

[25] Jules Bloomenthal and Keith Ferguson *Polygonizaiton of Non-Manifold Implicit Surface,* SIGGRAPH 1995.

[26] Luiz Henrique de Figueiredo, Jorge Stolfi, *Adaptive enumeration of implicit surfaces with affine arithmetic,* Implicit Surface 1995.

[27] Luiz Velho *Simple and Efficient Polygonization of Implicit Surface,* Journal of Graphics Tools, 1(24), pp.5-24, 1996.

[28] Andrea Bottino, Wim Nuij, Kees van Overveld, *How to Shrinkwrap through a Critical Point: an Algorithm for the Adaptive Triangulation of Iso-Surfaces with Arbitrary Topology,* Proceedings of IS'96 Conference, Eindhoven, September 1996.

[29] Luiz Henrique de Figueiredo, Jonas Gomes, *Sampling implicit objects with physically-based particle systems,* Compute & Graphics 20 #3, 1996

[30] Luiz Henrique de Figueiredo, *Surface intersection using affine arithmetic,* Graphics Interface 1996.

[31] Kazutoshi Yonekawa, Ken-ichi Komori, Toshiro Kutsuwa, *A Geometric Modeler by Using Spatial-Partitioning Representations*, IPSJ Vol.37 No.1 1996.

[32] Raj Shekhar, Elias Fayyad, Roni Yagel, J.Fredrick Cornhill *Octree-Based Decimation of Marching Cubes Surfaces*, Visualization 1996, pp335-342.

[33] Angela Rosch, Matthias Ruhl, Diemar Saupe, *Interactive Visualization of Implicit Surfaces with Singularities,* EuroGraphics Volume 16, pp. 295-306, 1997.

[34] Barton T. Stander, John C. Hart, *Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling,* SIGGRAPH 1997.

[35] Nilo Stolte, Arie Kaufman, *Discrete Implicit Surface Models using Interval Arithmetics,*
Second CGC Workshop on Computational Geometry, Durham, Duke University, October 1997.

[36] Nilo Stolte, Arie Kaufman, *Discrete Implicit Surface Models using Interval Arithmetic,* In Second CSG WorkShop, October 1997.

[37] Nilo Stolte, Rene Caubet, *Comparison between Different Rasterization Methods for Implicit Surfaces,* Visualization and Modeling, chapter 10, pages 191-201, April 1997.

[38] Hans-Christian Hege, Martin Seebab, Detlev Stalling, Malte zockler *A Generalized Marching Cubes Algorithm Based On Non-Binary Classifications,* 1997.

[39] Wolfgang Heidrich, Philipp Slusallek, Hans-Peter Seidel, *Sampling of Procedural Shadrs Using Affine Arithmetic,* ACM Transactions on Graphics (TOG) archive Volume 17, Issue 3, July 1998.

[40] Nilo Stolte, Arie Kaufman, *Parallel Spatial Enumeration of Implicit Surfaces Using Interval Arithmetic for Octree generation and its direct Visualization,* Implicit Surface 1998.

[41] Affonso de Cusatis Junior, Luiz Henrique de Figueiredo, Marcelo Gattass, *Interval Methods for Ray Casting Implicit Surfaces with Affine Arithmetic,* In processing SIBGRAPH pp.65-77, 1999.

[42] John C. Hart *Computational Topology for Shape Modeling,* in Proc. of Shape Modeling International 99, Aizu-Wakamatsu, Japan, March, 1999.

[43] John C. Hart, *Using the CW-Complex to Represent the Topological Structure of Implicit Surfaces and Solid,* SIGGRAPH 1999.

[44] Luiz Velho, Luiz Henrique de Figueiredo, Jonas Gomes, *A Unified Approach for Hierarchical Adaptive Tesselation of Surfaces,* ACM Graphics October 1999.

[45] R.Westermann, L. Kobbelt, T.Etrl *Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surface,* The Visual Computer, 15, pp.100-111, 1999

[46] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, V. Savchenko, HyperFun project: *A framework for collaborative multidimensional F-rep modeling,* Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop, J. Hughes and C. Schlick (Eds.), pp. 59-69.

[47] Adrian Bowyer, Jakob Berchtold, David Elisenthal, Irina Voiculescu, Kevin Wise, *Interval Methods in Geometric Modeling*, IEE Geometric Modeling and Processing 2000.

[48] Irina Voiculescu, Jakob Berchtold, Adrian Bowyer, Ralph R. Martin, and Qijiang Zhang *Interval and Affine Arithmetic for Surface Location of Power- and Bernstein-form Polynomials*, IEE Geometric Modeling and Processing 2000.

[49] Ronald N. Perry, Sarah F. Frisken *Kizamu: A System For Sculpting Digital Characters,* SIGGRAPH 2001, Technical Report 2001 TR2001-08

[50] Nilo Stolte, Arie Kaufman, *Robust Hierarchical voxel Models for representation and Interactive visualization of Implicit Surfaces in Spherical Interactive visualization of Implicit surfaces in Spherical Coordinates,* Graphical Models 2001.

[51] Tasso Karkanis, A.James Stewart, *Curvature-Dependent Triangulation of Implicit Surfaces* IEEE, 2001.

[52] Helio Lopes, Joao Batista Oliveira, Luiz Henrique De Figueiredo, *Robust Adaptive Approximation of Implicit Curbes*, XIV Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPHI'01) October 15 - 18, 2001.

[53] Katja Buehler, *Fast and Reliable Plotting of Implicit Curves*, Uncertainty in Geometric Computations; Joab Winkler, Mahesan Niranjan (eds.); Kluwer Academic Publishers. pp. 15-28, 2002

[54] Tasso Karkanis, A. James Stewrt, *High Quality, Curbature Dependent Triangulation of Implicit Surfaces,* IEEE Computer Graphics and Applications, 21(2):60-69, (March 2001).

[55] Xikun Liang, Brian Wyvill, *Hierarchical Implicit Surface Refinement,* Computer Graphics Internaional, 2001.

[56] Feif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, Hans-Peter Seidel p *Feature Sensitive Surface Extraction from Volume Data,* SIGGRAPH 2001

[57] Youichi Gotou, *Interactive Modling and Visualization of F-rep Solids with an Extendable User Interface,* Master thesis in University of Aizu, 2002.

[58] Greg Turk, James F. O'Brien, *Modelling with Implicit Surfaces that Interpolate,* ACM Transactions on Graphics, Vol. 21, No. 4, October 2002, Pages 855-873.

[59] Huahao Shou, Ralph Matrin, Irina Voiculescu, Adrian Bowyer, Goujin Wang, *Affine Arithmetic in Matrix Form for Polynomial Evaluation and Algebraic Curve Drawing,* Progress in Natural Science 12 (1): pp. 77-81 January 2002.

[60] Yu. Ohtake and A. G¿ Belyaev *Mesh optimization for polyonized isosurfaces,* Computer Graphics Forum(Eurographics 2001), 20(3):368-376, September 2001.

[61] Tao ju, Frank Losasso, Scott Schaefer, Joe Warren *Dual Contouring of Hermite Data,* SIGGRAPH 2002

[62] Scott Schaefer, Joe Warren *Dual Contouring:"The Secret Sauce",* Techinical Report TR 02-408

[63] Ralph Martin, Huahao Shou, Irina Voiculescu, Adrian Bowye, Guojin Wang *Comparison of interval methods for plotting algebraic curves ,* Computer Aided Geometric Design, Volume 19, Issue 7, July 2002, pp. 553-587.

[64] Adrian Bowyer, Ralph Martin, Huahao Shou, Irina Voiculescuk, *Affine intervals in a CSG geometric modeller,* Proc. Uncertainty in Geometric computations ISBN 0-7923-7309, pp. 1-14.

[65] Huahao Shou, Ralph Matrin, Irina Voiculescu, Adrian Bowyer, Goujin Wang, *Affine Arithmetic and Bernstein Hull Methods for Algebraic Curve Drawing,* `http://users.comlab.ox.ac.uk/irina.voiculescu/Publications/Sheffield_AA_paper_2001.pdf`, 2001.

[66] Katja Buhler, *Implicit Linear Interval Estimations,* Proceedings of the 18th Spring Conference in Computer Graphics (SCCG'02); Budmerice, Slovakia; ACM; 2002.

[67] Thomas Thebl, Torsten Moller, Meister Eduard Groller, *Optimal Regular Volume Sampling,* IEEE visuallization 2001.

[68] Yutaka Ohtake, Alexander Belyaev, Alexander Pasko, *Dynamic Mesh Optimization jfor Polygonized Implicit Surfaces with Sharp Features.* The Visual Computer 2002.

[69] K.Levinski, A.Sourin, *Interactive Polygonization for function-based shpae modelling,* Eurographics 2002.

[70] Luiz Henrique de Figueiredo, Jorge Stolfi, *Affine Arithmetic: Concepts and Applications,* Numerical Algorithm 00: 1-13, 2003.

[71] Hamish Carr, Thomas Theubl, Torsten Moller, *Isosurface on Optimal Regular Samples,* EUROGRAPHICS 2003.

[72] Luiz Henrique de Figueiredo, Jorge Stolfi, Luiz Velho, *Approximating Parametric Curves With Strip Trees Using Affine Arithmetic,* Eurographics volume 22 number 2 pp. 171-179, 2003.

[73] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, Hans-Peter Seidel, *Multi-level Partion of Unity Implicits,* SIGGRAPH 2003.

[74] Bart Adams, Philip Dutre *Interactive Boolean Operations on Surfel-Bounded Solids,*

[75] Mark Pauly, Richard Keiser, Leif P. Kobbelt, Markus Gross, *Sahep Modeling with Point-Sampled Geometry,* SIGGRAPH 2003.

[76] Milos Hassan, *An Efficient F-rep Visualizzation Framework,* M.Sc. thesis, Faculty of Mathematics, Physics and Informatics, Com enius University, Bratislava, Slovakia

[77] *Libaa,* http://www.nongnu.org/libaa/

[78] Jules Bloomenthal, Chandrajit Bajaj, Jim Blinn, Marie-Paule Cani-Gascuel, Alyn Rockwood, Brian Wyvill, Geof Wyvill, *Introduction to Implicit Surfaces,* Morgan Kaufmann, ISBN 1-55860-233-X, 1997.