UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Triangulation of Non-Manifold Implicit Surfaces using Geometric Healing Techniques

Quoc Trong Nguyen

Dissertation for obtaining the degree of Master of Science in
Computer Science and Engineering
(2nd Cycle of Studies)

Supervisor: Prof. Dr. Abel João Padrão Gomes

Covilhã, June of 2016

”Thanks be to God for his indescriable gift!”
(2 Corinthians 9:15 NIV)

Acknowledgements

I would like to express my gratitude to my family, Professor Abel Gomes, and Professor Nuno Garcia for their love, confidence, patience, and generousness.

I am also grateful to my brothers and sisters in Christ at the Southern Evangelical Church of Vietnam who have remembered me in their prayers.

I have also to thank my friends at the Department of Computer Science, University of Beira Interior for all of their help in studying and working: Daniel, Diana, Celina, Carlos, Paula, Virginie, Dmytro, Hugo, Tiago Simões, and Professor Nuno Pombo.

Resumo

As superfícies implícitas utilizadas em modelação geométrica estão muitas vezes limitadas a variedades topológicas bidimensionais (*2-dimensional manifolds*, do inglês) sem singularidades, porque são definidas como funções de conjuntos zero que separam o espaço em regiões binárias. Mas, em geral, as superfícies implícitas apresentam singularidades tais como pontos isolados e linhas de auto-interseção, pelo que são essencialmente não-trianguláveis. Isto significa que a triangulação e renderização de tais superfícies não são tarefas triviais.

Esta dissertação descreve a conceção e a implementação de um algoritmo para a triangulação e renderização de superfícies implícitas com singularidades, utilizando para isso técnicas de restauração ou cura geométrica. O algoritmo apresentado, designado por *healed marching cubes* (HMC), pode ser visto como uma extensão do algoritmo *marching cubes* (MC), introduzido por Lorensen e Cline em 1987 [LC87].

No contexto de triangulação e renderização de superfícies implícitas, não existem muitos algoritmos propostos na literatura para resolver os seus problemas típicos, como a resolução de auto-interseções em superfícies com singularidades. Por isso, o algoritmo de *healed marching cubes* (HMC) poderá ser uma contribuição valiosa para a área de computação gráfica e computação geométrica.

Palavras-chave

Superfície implícita

Variedade topológica

Variedade topológica com auto-interseções

Auto-interseções

Triangualação de superfícies

Marching cubes

Inserir palavras-chave (Keywords in Portuguese)

Abstract

Implicit surfaces used in geometric modeling are often limited to two-dimensional manifolds because they are defined as zero-set functions that separate the space into binary regions. Non-manifold implicit surfaces containing singularities such as isolated points or self-intersection points are essentially non-polygonizable. Thus, triangulating and rendering such surfaces are not a trival task.

This dissertation presents the design and implementation of an algorithm for triangulating and rendering of non-manifold implicit surfaces using geometric healing techniques. The presented algorithm, called Healed Marching-Cubes (HMC), is built on the standard Marching-Cubes algorithm introduced by Lorensen and Cline in 1987 [LC87].

In the context of implicit surface triangulation and rendering, there are not many proposed algorithms to solve its typical problems such as self-intersection emerged by non-manifold surfaces. Hence, Healed Marching-Cubes will be a valuable contribution to the field of computer graphics and geometric computing.

Keywords

Implicit surface

Manifold

Non-manifold

Self-intersection

Surface triangulation

Marching-Cubes

Contents

List of Figures

List of Tables

List of Acronyms

MC	Marching-Cubes
HMC	Healed Marching-Cubes
MT	Marching-Tetrahedra

Chapter 1

Introduction

This chapter aims at introducing the problem addressed in this dissertation, as well as the research field of numerical computational geometry, as known as geometric modeling or computer-aided design. The research problem here approached has to do with the triangulation of non-manifold implicit surfaces by extending the standard marching cubes algorithm with a geometric healing technique inspired in differential geometry.

1.1 Motivation

Implicit surface triangulation is an active research topic in geometric modeling and computer graphics. Its importance is noticeable in the commercial and free modeling tools for the CAD/CAM applications [HF08]. Besides, as a part of surface modeling techniques, it has been applied in many other applications, mainly in medical imaging, molecular modeling, just to name a few.

In computer graphics, implicitly-defined surfaces play a fundamental role because it is a very convenient way to define surfaces, or more generally, isosurfaces [BW97, VGdF07, TALSZ14]. Recently, there is an increasingly trend in using implicits to reconstructing surfaces from different types of data, in particular those acquired from scanning devices. This is so because surfaces generated from point clouds are smooth, though it is also admissible to have sharp features like, for example, apices and creases, wit the advantage that it is not required a special treatment for topology changes [WF07, BCSV04].

Traditionally, implicitly-defined surfaces are two-dimensional manifold surfaces [BF95, HF08], for which there is a significant number of triangulation algorithms [GVJ+09]. For example, the marching cubes algorithm is a well-known triangulation algorithm for manifold surfaces [LC87].

It happens that most triangulation algorithms found in the literature assume that implicitly-defined surfaces are 2-dimensional manifolds. These polygonizers are not prepared to deal with non-manifold implicit surfaces, i.e., surfaces with singularities like self-intersections and isolated points. In the literature, we find only a few algorithms capable of dealing with non-manifold implicit surfaces (see [RG06] for a representative example). The purpose of this thesis is thus to propose a triangulation algorithm for non-manifold implicit surfaces.

1.2 Problem Statement

Before proceeding any further, let us have a refresher on implicit surfaces. An implicit surface S can be defined as the zero set of a real function $f : \mathbb{R}^3 \to \mathbb{R}$, i.e.,

$$S = \{\mathbf{p} \in \mathbb{R}^3 : f(\mathbf{p}) = 0\} \tag{1.1}$$

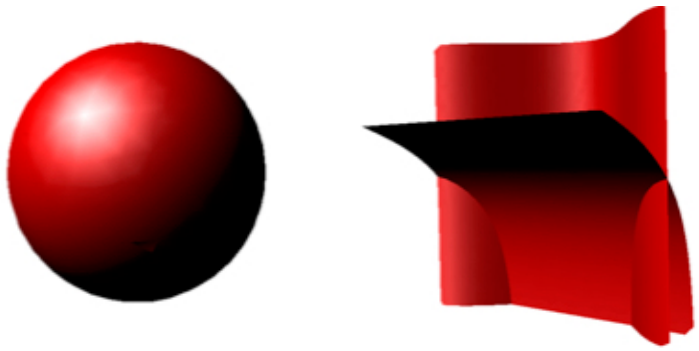
Interestingly, the function f is also known as implicit function, level-set function, scalar-valued function, field distance function, signed distance function, smooth function, or point-membership classification function, what is clearly dependent on the purpose or application at hand.

The geometric representation of the implicit function f is known as isosurface. For example, a family of spheres centered at the origin can be defined by the following implicit function:

$$f(\mathbf{p}) = x^2 + y^2 + z^2 - c^2 \tag{1.2}$$

with $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ and $c \in \mathbb{R}^+$. For example, the unit sphere can be represented by the following zero set of points:

$$B = \{\mathbf{p} \in \mathbb{R}^3 : f(\mathbf{p}) = 0 \quad \text{and} \quad c = 1\} \tag{1.3}$$



(a) (b)

Figure 1.1: Implicit surfaces (pictures taken from [RG06]): (a) the manifold surface $f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$; (b) the non-manifold surface $f(x, y, z) = x \ln x + \ln x \cos z - xy - y \cos z = 0$.

The leading idea of the algorithm proposed in this thesis is to able to triangulate bon manifold and non-manifold implicit surfaces. An example of a manifold surface is the sphere shown in Figure 1.1(a), while Figure 1.1(b) depicts a non-manifold surface, which as two self-intersection curved lines.

The problem tackled in this thesis has mainly to do with the triangulation of implicit surfaces, no matter they manifold or not. This means that such an algorithm must preserve the topology of any surface, even when it exhibits singularities. It happens

that it is widely known that the marching cubes algorithm do not offer guarantees in topological terms, i.e., singularities are simply ignored in the triangulation. So, the problem statement that central to this thesis is as follows:

> *Is it feasible to reformulate the marching cubes algorithm in order to cope with singularities?*

As will be shown throughout the thesis, it is possible to triangulate a non-manifold implicit surface correctly, since we are capable of determining its singularities beforehand, from which one proceeds to the re-triangulation of cubes where we find singularities of the surface.

1.3 Publications

In the course of the work underlying this thesis, two scientific articles have been produced as follows:

Quoc T. Nguyen and Abel J.P. Gomes. 2016. A Marching Cubes Triangulation Algorithm for Non-Manifold Implicit Surfaces. *IEEE Transactions on Visualization and Computer Graphics* (to be submitted), 2016.

Abstract. No many algorithms have been developed for non-manifold surfaces so far, largely because of the difficulties in resolving singularities, either symbolically or numerically. In this dissertation, we manage to resolve such singularities using a numerical strategy. This numerical approach allows us to extend the traditional marching cubes triangulation algorithm to non-manifold implicit surfaces with self-intersections. Such an extended algorithm is here called *healed marching cubes* (HMC).

Sérgio E. D. Dias, Quoc T. Nguyen, Joaquim A. Jorge, and Abel J.P. Gomes. 2016. Multi-GPU-Based Detection of Protein Cavities using Critical Points. *Future Generation Computer Systems* (submitted after minor revisions), 2016.

Abstract. Protein cavities are specific regions on the protein surface where ligands (small molecules) may bind. Such cavities are putative binding sites of proteins for ligands. Usually, cavities correspond to voids, pockets, and depressions of molecular surfaces. The location of such cavities is important to better understand protein functions, as needed in, for example, structure-based drug design. This article introduces a geometric method to detecting cavities on the molecular surface based on the theory of critical points. The method, called CriticalFinder, differs from other surface-based methods found in the literature because it directly uses the curvature of the scalar field (or function) that represents the molecular surface, instead of evaluating the curvature of the Connolly function over the molecular surface. To evaluate the accuracy of CriticalFinder, we compare it to other seven geometric methods (i.e., LIGSITE^{CS}, GHECOM, ConCavity, POCASA, SURFNET, PASS, and Fpocket). The benchmark results

show that CriticalFinder outperforms those methods in terms of accuracy. In addition, the performance analysis of the GPU implementation of CriticalFinder in terms of time consumption and memory space occupancy was carried out.

1.4 Dissertation Organization

This thesis has been organized as a regular dissertation, not as a series of published papers. To be more specific, this thesis is organized as follows:

Chapter 1. This chapter introduces the thesis itself, the problem statement, as well as the motivation that has led to the endeavor of writing this thesis.

Chapter 2. This chapter approaches the background necessary to better understand the details of implicit surfaces.

Chapter 3. This chapter reviews the most relevant triangulation algorithms that make use of space partitioning schemes.

Chapter 4. This chapter details how the marching cubes have been re-designed in order to triangulate non-manifold implicit surfaces.

Chapter 5. This chapter concludes the thesis, though a few open issues have been approached for future work.

Furthermore, there is also an appendix and a glossary in the end of thesis. The glossary contains the important terms used in this dissertation.

Chapter 2

Fundamentals of Implicit Surfaces

This chapter introduces the mathematical fundamentals underlying implicit surfaces in 3-dimensional space. Starting from the definition of implicit function put forward on the previous chapter, we will pay attention to the properties of the implicit representation for surfaces. In particular, the notions of manifold and non-manifold surfaces will be clarified. Moreover, an overview on triangulations of implicit surfaces will be addressed in the context of this dissertation.

2.1 Surface Representations

There are three main forms to define a surface in $\mathbb{R}^3$: *explicit*, *parametric*, and *implicit*.

The plane $A\,x + B\,y + C\,z + D = 0$ takes its explicit form as $z = -\frac{1}{C}(A\,x + B\,y + D)$, so that the value of $z = f(x, y)$, with $f(x, y) = -\frac{1}{C}(A\,x + B\,y + D)$, depends on the values of x and y. That is, the *explicit form* of a surface is given by $z = f(x, y)$. However, finding an explicit form for a surface is hardly feasible, mainly for analytic expressions of degree greater than 2.

For example, the torus surface is *parametrically* defined as

$$\begin{aligned} x(u, v) &= \cos(u)\,[R + r\cos(v)] \\ y(u, v) &= \sin(u)\,[R + r\cos(v)] \\ z(u, v) &= r\sin(v) \end{aligned} \tag{2.1}$$

where r and R stand for the minor and major radii, respectively; the angle parameters u and v both vary in the range $[0, 2\pi]$. Therefore, a parametric surface in $\mathbb{R}^3$ is defined by a vector function $f : \mathbb{R}^2 \to \mathbb{R}^3$, i.e., $f(u, v) = \langle x(u, v), y(u, v), z(u, v)\rangle$.

On the other hand, an implicit surface is defined as the zero set of a real function $f : \mathbb{R}^3 \to \mathbb{R}$. For example, the unit sphere is implicitly defined as:

$$f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0 \tag{2.2}$$

This function f is also known as implicit function, level-set function, scalar-valued function, scalar function, field distance function, signed distance function, or point-membership classification function. Different ways of name calling have different purposes of using the implicitly-defined function. The geometric representation of such function f is called as an isosurface.

2.2 Point Membership

The implicit representation of a surface provides an easy and direct way to test whether a point $(x, y, z) \in \mathbb{R}^3$ is on the surface or not, simply by checking the value of $f(x, y, z)$. For example, let us consider the unit sphere defined as the zero set of $f(x, y, z) = x^2 + y^2 + z^2 - 1$; so, $f(x, y, z) = 0$ for points of the sphere itself, $f(x, y, z) > 0$ for points outside the sphere, and $f(x, y, z) < 0$ for points inside the sphere. This procedure of checking whether a point belongs to a surface or not is known as *point-membership* test. This is an important feature of implicitly-defined surfaces that is leveraged in many applications in computer graphics, in particular in their triangulations.

However, unlike parametric surfaces, there is no direct way to systematically generate a set of points on implicit surfaces. This is the reason why this type of representation is called "implicit" [GVJ+09]. Operations on implicit surface are strongly supportted by the implicit function theorem, which sets the conditions under which the implicit function can be solved (theoretically) for each point of the surface. But, in general, the solution may not exist, particularly at singular points.

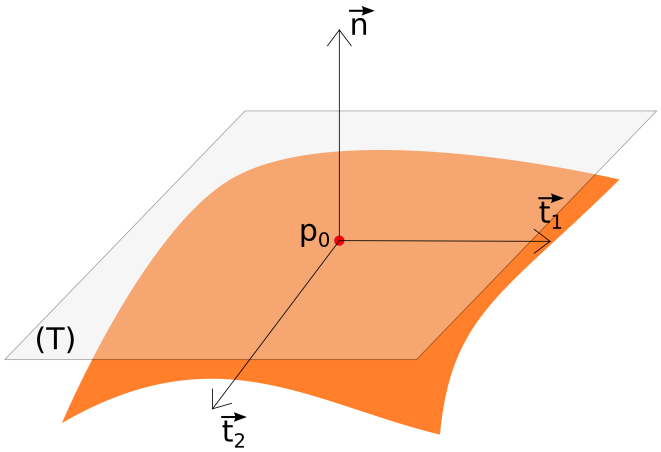


Figure 2.1: Tangent plane and normal vector at the point p_0 on the surface (in orange).

2.3 Gradient, Normal Vector, and Tangent Plane

Let $f : S \subset \mathbb{R}^3 \to \mathbb{R}$ be a real-valued function that defines an implicit surface S as its zero set . The surface S is said to be smooth if the gradient vector $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})$ is continuous and non-zero at any point of S. Equivalently, we can write:

$$\nabla f = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j} + \frac{\partial f}{\partial z}\vec{k} \neq (0, 0, 0) \tag{2.3}$$

where $\vec{i} = (1, 0, 0)$, $\vec{j} = (0, 1, 0)$, $\vec{k} = (0, 0, 1)$ are the standard unit vectors.

Then, let consider a specific point $(x_0, y_0, z_0) \in S$, the gradient ∇f at (x_0, y_0, z_0) is

a vector perpendicular to the surface S, here also represented by $\vec{n}$, as illustrated in Figure 2.1.

Besides, the equation of the tangent plane (T) at point (x_0, y_0, z_0) of the surface S is given by the following equation:

$$\frac{\partial f}{\partial x}(x - x_0) + \frac{\partial f}{\partial y}(y - y_0) + \frac{\partial f}{\partial z}(z - z_0) = 0 \tag{2.4}$$

where the partial derivatives are obviously calculated at (x_0, y_0, z_0). As shown in Figure 2.1, the triple $(\vec{t_1}, \vec{t_2}, \vec{n})$ forms a local system of coordinates at (x_0, y_0, z_0) on the surface, where $\vec{t_1}$ and $\vec{t_2}$ are orthonormal in the tangent plane.

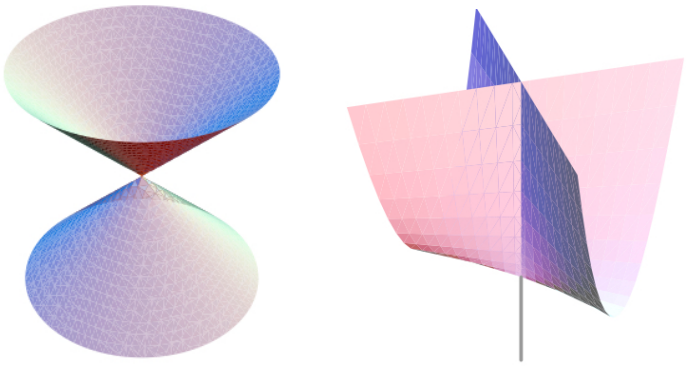
2.4 Singular Points

A point (x, y, z) on the implicit surface is called singular (or critical) if the gradient vector at such a point vanishes. In other words, a surface point is said to be a singular point if its partial derivatives vanish simultaneously, i.e.,

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0 \tag{2.5}$$

Singular points (or singularities) are part of the self-intersection lines or sharp edges on the surface. An implicit surface possessing singular points is called a non-smooth implicit surface; otherwise, it is called smooth implicit surface.

Generally, singularities can be classified into two main types: 0-dimensional singularities (i.e, isolated points and touching points) and 1-dimensional singularities (i.e., self-intersection lines) [Mor03, GDM10]. In Figure 2.2(a), the apex of the double cone is a touching point, while the surface in Figure 2.2(b) exhibits a self-intersection straight line that coincides with the z-axis.



(a) (b)

Figure 2.2: Non-smooth implicit surfaces (pictures taken from [GVJ+09]): (a) a double cone defined as the zero set of the implicit function $f(x, y, z) = x^2 + y^2 - z^2$; (b) a cross-cap defined as the zero set of the implicit function $f(x, y, z) = x^2 y - z^2$.

2.5 Manifold Implicit Surfaces

A manifold implicit surface is a surface defined in the 3-dimensional Euclidean space that enjoys the following condition; the neighborhood of every single point on the surface is topologically the same as the open unit disc (i.e., a 2-dimensional disc). This is sometimes called the *manifoldness condition*. In other words, such neighborhood looks like R^2 locally. In general, any implicit surface that is nearly "flat" on small scales is a manifold surface; for example, a toroidal surface is a manifold surface, so do the surface of a cube, even with those corners and creases. In short, a manifold surface admits single-sheet singularities, but not singularities as those shown in Figure 2.2.

2.6 Non-Manifold Implicit Surfaces

Non-manifold implicit surfaces are surfaces that do not satisfy the manifoldness condition as stated above. Therefore, those two surfaces depicted in Figure 2.2 are non-manifold surfaces because the neighborhood of at least one surface point contains more than one sheet (or 2-dimensional disc) of the surface itself. This thesis is much about detecting these singularities. More examples of non-manifold surfaces are shown in Figure 2.3

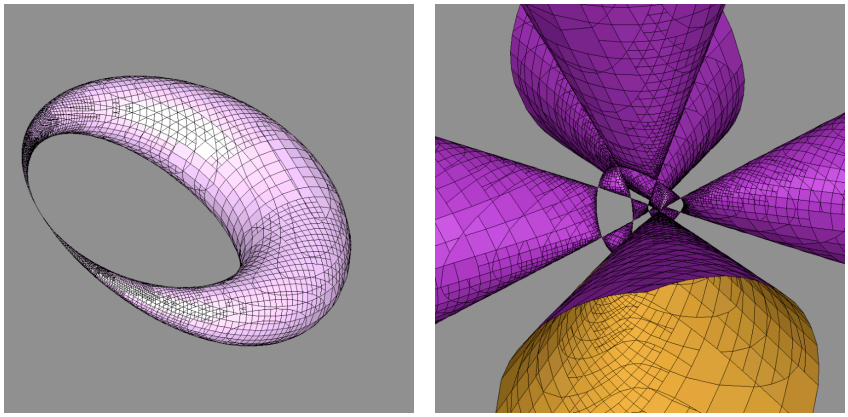


Figure 2.3: Non-manifold surfaces containing touching points (pictures abusively taken from [BS05]).

2.7 Triangle Meshes

Triangle meshes are the most widely used representation of shapes in computer graphics because of their algorithmic simplicity, numerical robustness, and efficient display [BPK+07, HVFF13, Ju09, KB04]. Triangle meshes consist of many triangles connected by common edges to form a triangulated surface that is well suited for rendering through graphics hardware [RRS97]. Some examples of triangulated surfaces are depicted in Figure 2.4.

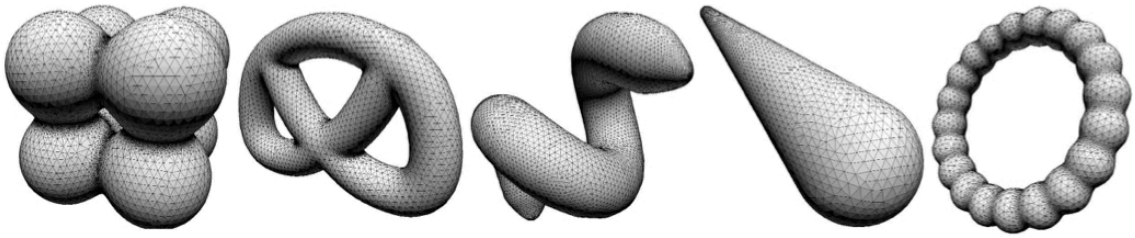


Figure 2.4: Examples of triangulated implicit surfaces (picture abusively taken from Karkanis and Stewart [KS01]).

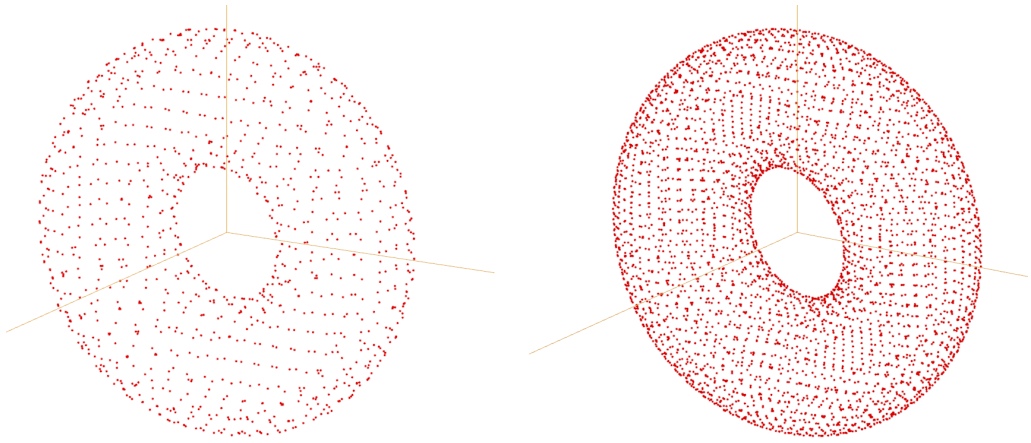
In topological terms, a triangle mesh is a connected set of disjoint vertices, edges, triangles. Therefore, it consists of three sets of building blocks, i.e., the set $V = \{v_1, ..., v_n\}$ of vertices, the set $E = \{e_1, ..., e_k\}$ of edges, with $e_i \in V \times V$, and the set $F = \{f_1, ..., f_m\}$ of faces, with $f_i \in V \times V \times V$. In geometric terms, a triangle mesh is defined by its vertex positions or points $P = \{p_1, ..., p_n\}$, with $p_i \in R^3$. This association between the topological mesh and the geometry of its vertices (i.e., their Cartesian positions) allows us to well build up a triangle mesh that approximates an implicit surface.

It is necessary to bear in mind that triangulating a manifold implicit surface is by far easier than a non-manifold implicit surface. We can even to say that triangulating a non-manifold implicit surface is not a trivial task indeed. In fact, the manifoldness condition means that every single edge of a triangle mesh has two incident triangles at maximum. In non-manifold implicit surfaces meshes, we have to ensure that self-intersections are represented by edges with three or more incident triangles; this is the case of the triangulation shown in Figure 2.2(b).

2.8 Implicit Surface Sampling

Triangle meshes are constructed through a process called triangulation (or, more generally, polygonization), in which polygons used to represent a surface are triangles. In this triangulation process, the set of sampled points are connected in a manner that the constructed triangle mesh approximates the surface correctly in term of geometry and topology. For example, Figure 2.5 shows two sets of points on the torus sampled with different densities; the toroidal surface is defined by the function $f(x, y, z) = (x^2 + y^2 + z^2 + \frac{3}{4})^2 - 4(x^2 + y^2)$.

Sampling is thus an important stage in the implicit surface triangulation process. The sampling stage, also known as the surface discretization, occurs before the triangulation stage. The purpose of this stage is to determine the set of points on the surface from which the triangle mesh representing the surface is constructed. The density of the point set (i.e., the number of points extracted from the surface) depends on the sampling resolution that is specified in the beginning of the triangulation process. High density means a more detailed triangulated surface in the end.



(a) (b)

Figure 2.5: Two sets of points of the toroidal surface sampled using a 2-point method for each edge of a grid of cubes: (a) sampling with grid resolution=0.15; (b) sampling with grid resolution=0.08.

From the real-valued function defining the implicit surface, the sampling is carried out within a limited domain (i.e., the space in which the sampling happens). Root-finding numerical algorithms are often applied in the context of sampling the implicit surfaces. In general, there are essentially two broad classes of numerical methods [GVJ+09]:

- 1-point methods: starting from a single point, this method calculates one point after another in a sequence that converges to the solution point within a small tolerance.
- 2-point methods: starting from interval that contains the root, this method attempts to shrink such interval until finding the the solution point within a small tolerance.

Each method has its own advantages and disadvantages. While most of 2-point methods are reliable, but slow, 1-points methods tend to be faster, but do not guarantee convergence. Furthermore, there is no certain method to determine the exactly appropriate density of point set to guarantee that all information on the surface (i.e., topological information) are extracted for all implicit surfaces. Besides, the geometric accuracy of the triangulated surface is always limited by the numerical errors in calculating with floating numbers.

In computer graphics, most algorithms for sampling implicit surfaces use a 2-point method. If the real-valued function defining the surface evaluates positive at the first point and negative at the second point, one can say that the surface is located somewhere between them. However, 2-point methods may fail to detect and sample the surface because the functions of many implicit surfaces evaluate either positive or negative everywhere around them. These surfaces are here called sign-invariant implicit surfaces [RG06]. In those cases, 1-point methods are the right alternative for sampling implicit surfaces.

2.9 Concluding Remarks

This chapter has introduced the fundamentals of implicit surfaces and the most relevant mathematical tools to succeed in sampling and triangulating implicit surfaces, no matter they manifold or not. In this dissertation, we are interested in non-manifold implicit surfaces, in particular those with self-intersections. For that purpose, we will take advantage of the standard marching cubes algorithm, extending it to non-manifold surfaces. As explained later in this dissertation, such an extension requires a numerical procedure to resolve the singularities of implicit surfaces. But, before proceeding any further, let us briefly review the relevant literature in the next chapter.

Chapter 3

Literature Review

This chapter presents a revision of the literature on the subject. Starting with an overview of the main classes of implicit surface triangulation techniques, the focus is then moved to the Spatial Partitioning class which Healed Marching-Cubes belongs to. Follow through is a revision of the healing algorithms for triangulating and rendering non-manifold implicit surfaces.

3.1 Introduction

In the context of implicit surface triangulation and rendering, many algorithms have been proposed to solve different aspects of the problem. In spite of the many proposed algorithms, they are developed based on the same main ideas. The foremost focus is how to sampling the implicit surface as accurately as possible. This is the necessary condition. After that, the question is how to triangulate the sampled points by connecting them correctly. This is the sufficient condition.

Mathematically, the implicit surface triangulation can be described as follows: given an implicit surface represented by a zero-set function $f(x, y, z) = 0$, one have to determine the set of points $S = (x_0, y_0, z_0) \in \mathbb{R}^3 : f(x_0, y_0, z_0) = 0$. Then, one proceeds to choose an appropriate strategy to connect the points in S together in order to obtain a formal form of function f. All the points in S are called sampled points, and the number of points in S is called the sampled density (or surface resolution, in computer graphics).

Generally, surface trianglution and rendering algorithms include three main steps:

$$\text{sampling} \rightarrow \text{triangulating} \rightarrow \text{rendering}$$

Sampling is an important step in surface triangulation. It ensures that the surface is approximated correctly in term of geometry. On the other hand, triangulation is required to connect the adjacent sampled points correctly to form an accurate triangular mesh representing the implicit surface.

Obtaining a precisely triangulated surface that covering the surface f is difficult because the surface is triangulated only through a finite set of sampled points. In general, as the sampling density increases, the triangulated surface is more likely topologically correct and converges to the original surface f. However, if the sampled points do not satisfy certain properties required by the algorithm, the triangulation program produces incorrect results [FO03]. In fact, there is always a tradeoff between precision and

processing time in the surface triangulation on computer. Thus, in order to obtain the possibly best triangulated surface, one must choose a time-consuming approach.

In the next sections, after identifying main classes of triangulating and rendering algorithms, the pioneer works of Spatial partitioning method are introduced. Those works are the fundamental algorithms on which many other approaches have been built.

At this point, it is worth to emphasize that, although there are many triangulation and rendering techniques found in the literature, the number of techniques for non-manifold implicit surfaces is not that many. One of the main reason, as mentioned by Gomes et al. [GDM10], is because non-smooth surfaces are essentially non-polygonizable.

3.2 Taxonomy of algorithms

Surface reconstruction algorithms found in the literature for implicit surfaces can be classified into three main classes [RG06, GVJ+09, GDM10, DALJ+15]:

1. *Spatial partitioning methods*: These algorithms subdivide (either regularly or adaptively) the space into a latice of cells to find those that intersect the implicit surface. Usually, cells are either cubes or tetrahedra. The sign of the describing function at the cell vertices determines a configuration type that guides the polygonization of the surface. Unlike cubes, tetrahedragenerate topologically consistent triangular meshes (i.e. without ambiguities), yet with distorted triangles. These distorted triangles requires some kind of post-processing procedure to repair the resulting mesh. On the other hand, the cubic cell-based polygonization may lead to ambiguous configurations as more than one mesh can be created for the same configuration type. Some disambiguation strategies have been proposed in the literature, including simplex decomposition, modified look-up table disambiguation, gradient consistency heuristics and quadratic fit, tri-linear interpolation techniques, and recursive subdivision of space into smaller subcells.

2. *Continuation methods*: Also known as Surface tracking techniques, they iteratively create a polygonal approximation of the surface by using a mesh growing scheme from a starting seed element on the surface. This seed element results from the intersection between the seed cell and the surface. Neighbor surface elements are found by intersecting the neighbor cells with the surface, i.e. the mesh growth results from the cells straddling the surface. Unfortunately, the algorithm may miss important shape details of the surface, including very small components of the surface or even isolated points, because cells are of constant size. Another drawback of this technique comes from the need of having a seed cell for each surface component, which may be a rather difficult requirement to fill in. Besides, other tracking techniques for tilling implicit surfaces include: predictor-corrector and piecewise-linear techniques.

3. *Surface fitting methods*: Unlike spatial partitioning techniques, surface fitting methods are not based on partitioning space into cells. Starting from a seed mesh that roughly approximates the implicit surfaces, these techniques progressively adapt and deform the current mesh towards the implicit surface. The main problem comes from the difficulty in attaching triangles patches together, which sometimes results in cracks or even dangling triangles in the tesselation.

These three fundamental approaches diverge on the way in which they sample the space and rendering the implicit surfaces.

In this thesis, the author mainly concerns with the indirect approach to visualize the implicit surface that is to use mesh to represent the surface.

3.3 Spatial partitioning algorithms

This section presents typical algorithms of spatial partitioning class for reconstructing and rendering implicit surfaces. Typically, these algorithms start with a decomposition of the space domain (e.g., bounding-box) into smaller subdomains or cells (e.g., cubic boxes) for sampling the space. The surface is then polygonised by one or more polygons within each cell that intersects the surface. At the final step, the polygonised pieces of the surface is rendered to be shown on the screen.

Early spatial partitioning algorithms were developed by Wyvill et al. [WKS86] for rendering soft and blobby objects, and Lorensen and Cline [LC87] who designed the Marching-Cubes algorithm for generating human organ surfaces from medical image data sets.

Three common types of spatial partitioning algorithms are [GVJ+09]:

- *Spatial exhaustive enumeration*: This type of algorithms partitions the space into axis-aligned cells (also called voxels). The most significant representative of this type is the well-known Marching-Cubes algorithms. Spatial exhaustive enumeration algorithms can be used to extract human organ surfaces from a pack of 2D pixel images. It is also applied to general trilinear surfaces implicitly defined by level sets.

- *Spatial continuation*: This is a hybrid scheme that combines exhaustive enumeration and continuation.

- *Spatial subdivision*: Subdivision is an adaptive space partitioning technique. It is another attemp to solve the ambiguity problems resulting from the use of regular space grids.

It follows the details of each spatial partitioning type with its typical algorithms.

3.3.1 Spatial exhaustive enumeration

3.3.1.1 Marching-Cubes algorithm

Marching-Cubes (MC) algorithm is a 3D isosurfacing algorithm that widely used in scientific visualization [GVJ+09, NY06, CX14, MSS94]. Its applications can be found in medical imaging, bioinformatics, geographical information systems, weather forecasting, and many others [GVJ+09]. MC is probably the most popular isosurfacing algorithm from which many new researches and developments have been grown up [NY06, DALJ+15].

MC algorithm was introduced by Lorensen and Cline [LC87] in the context of medical imaging, though a similar algorithm due to Wyvill et al. [WMW86] has published before in the context of modelling soft objects. The main difference between the two algorithms lies in their spatial indexing data structures. The first uses a voxel-based data structure (i.e. a 3D array that mimics the partitioning of the bounding box into cubes), while the second uses a hash-table structure [GVJ+09].

When applied on implicit surfaces, Marching-Cubes extracts a triangulated iso-surface from a 3D rectilinear grid of uniformly sampled data values [CX14]. MC walks through three stages to reconstruct an implicit surface expressed by an implicit function:

- Partitioning of the bounding-box
- Sampling the surface
- Triangulation of the surface

The first stage partitions the bounding-box into a grid of cubes. The size of each cube (or the number of cubes in the grid) is decided by the size of the bounding-box and a user-desired resolution. After this stage, the bounding-box is composed of an axis-aligned grid of equally sized cubes. In fact, this discretisation of the bounding box need not to be done explicitly. It is enough to store the vertex data of each cube into a n-dimensional array, where n is the dimension of the cube (or of the space where the level set lies in).

MC constructs a facetized isosurface by processing each cube at a time. After having processed one cube, it moves (marches) to the next cube in a sequential order. The simplicity of MC comes from this cube-by-cube processing manner resulted in one-to-one mapping between the cubes created inside the bounding-box and the elements of the array data structure.

The second stage concerns the sampling of the implicit surface. This is the critical part of the algorithm whereby all necessary points for constructing the surface are determined. This step involves a sequence of three processes as mentioned in [GVJ+09]:

$$\text{evaluation} \rightarrow \text{classification} \rightarrow \text{interpolation}$$

In the evaluation process, the function value at each vertex of the current cube is estimated and stored in the corresponding data structures for vertices. A function value is either positive or negative presenting a binary state at a vertex. With this covention, the topological configuration of the surface within the cube is encoded, a bit per vertex, basing on estimated function values. If a vertex has a function value equal or less than isovalue c of the surface, its bit is set to 0, otherwise, it is set as 1.

The classification process uses the bit encoding of vertices to recognize the topological configuration of the surface in each cube.

Since there are two states at each cube vertex and a 3-cube posseses 8 vertices, there are $2^8 = 256$ possible topological configurations of the implicit surface in a cube. A look-up table is created to contain the edges intersecting the surface for each case. In practice, using cube symmetric properties (reflections and rotations), those 256 possible topological configurations are reduced to fourteen unique cases (Figure 3.1). A cube edge crosses the surface if its vertices have distinct values (0 and 1).

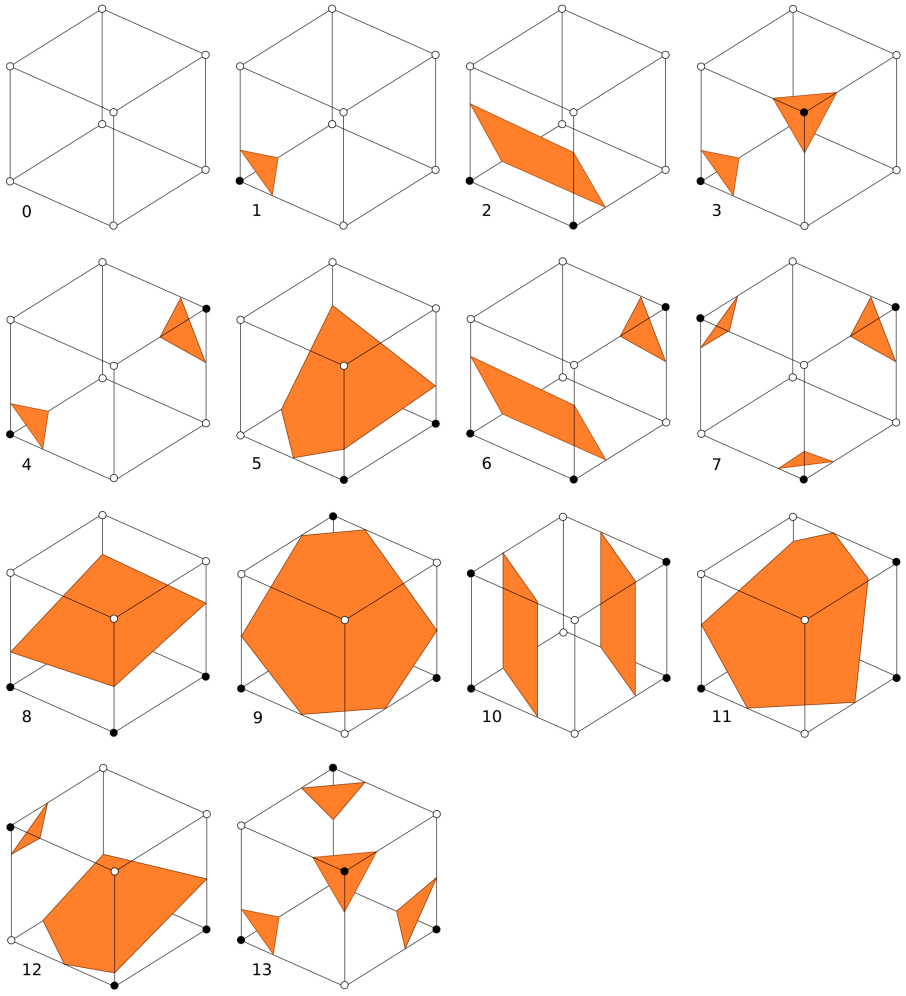


Figure 3.1: Topological configurations for marching cubes.

In the interpolation process, basing on the topological configuration of the cubes, the surface-edge intersection points in active cubes (cubes that intersect the implicit sur-

face) are interpolated. The intersection points usually are determiend by numerical interpolation (i.e., 2-point numerical methods), which may fail without using those certified techniques (e.g., interval arithmetic).

Having found the intersection points, the last stage in MC is to generate triangular facets that represent the portion of the isosurface that intersects each cube. The intersection points define the vertices of the triangles, and the collection of the triangular facets across all the cubes forms the triangular mesh that define the surface. The facetization pattern in each cube can be determined from the intersection topology look-up table [NY06].

Advantages

Part of its appeal is that it follows a straightforward, practical approach. MC can be modified to be used for many purposes. Practically, MC has been extended in a number of ways [NY06]. Besides, by leaveraging the hash-table structure, MC has gained a combination of simplicity, efficiency, and high speed.

Disadvantages

Same as any other algorithms, MC also gets several certain challenges. One disadvantage of isosurfaces created by MC is that they can exhibit visible faceting artifacts. Use of a higher-degree isosurface representation is one means to reduce these artifacts. MC guarantees neither correctness nor topological consistency. MC can also produce a topologically inconsistent isosurface that contains holes caused by one type of facetization ambiguity. Another type of ambiguity produces topologically consistent but incorrect isosurfaces [NY06].

Ambiguities

Marching-Cubes algorithm does not offer topological guarantees. Dürst [Dür88] discovered that some of the basic intersection topologies actually could be facetized in multiple ways. The authors of [VW94, Che95, Nat94] have demonstrated that seven topologies are ambiguous-Cases 3, 4, 6, 7, 10, 12, and 13.

The first type of ambiguities is the face ambiguities. In Figure 3.2, two adjacent cubes that share a face are shown. Each of those cubes contains a piece of surface that is polygonised following the MC patterns. However, the shared face is intersected differently in each cube. This difference arises because there are multiple possible cube-surface intersection patterns in both cubes, and the default intersection patterns are inconsistent on the shared face. The unresolved ambiguity produces a hole in the isosurface [NY06].

Another type of ambiguities is the internal abiguity (see Figure 3.3). Natarajan [Nat94] was the first to identify the internal ambiguity problem. Various techniques have been proposed to overcome shape ambiguity problems on the boundary and interior of Marching-Cubes. Two of these algorithms are the Dividing-Cubes and the Marching-Tetrahedra, which can be viewed as variants of the Marching-Cubes.

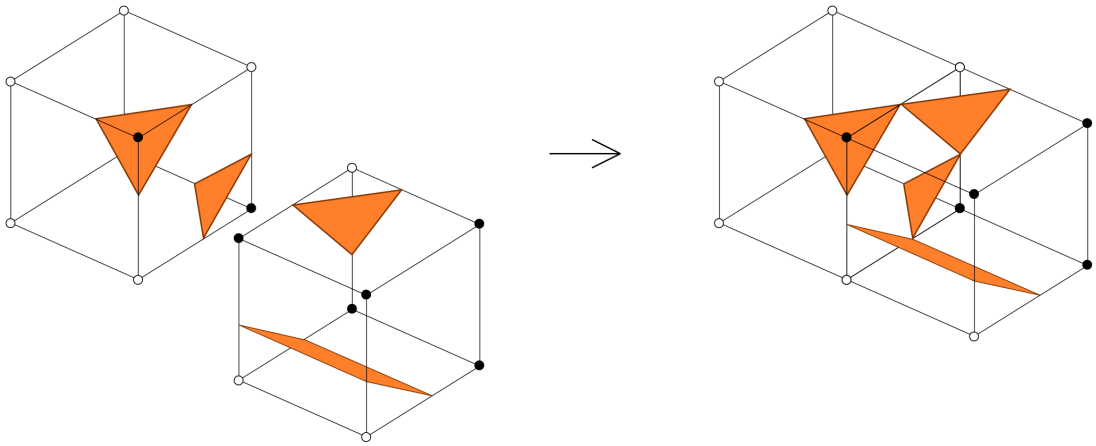


Figure 3.2: Marching cubes' face ambiguity problem.

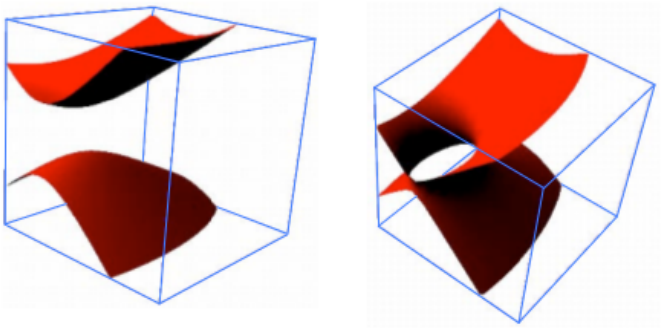


Figure 3.3: Marching cubes' internal ambiguity problem [GVJ+09].

3.3.1.2 Dividing-Cubes

Dividing-Cubes algorithm was proposed by Cline et al. [CLL+88] and is a variant of Marching-Cubes. Dividing cubes algorithm was developed to eliminate the scan conversion step of polygonal display algorithm. In medical images there are many surface elements and the triangles created by the polygonal algorithm may occupy a small area on the raster display. As the number of facet increases, the size of each triangle decreases and approaches the pixel size. For complex medical images, it is more efficient in memory and time to display point primitives on the raster display directly [Hu98].

Chernyaev [Che95] has observed that Dividing-Cubes [CLL+88]) is free from internal ambiguity. Dividing cubes differs from Marching-Cubes in that each cube is divided into pixel-sized cubes (also called pixel-sized voxels) that lie on the surface, and projects the intensity calculated for each cube onto the viewing plane. This division depends on both image and data resolution.

In the sampling stage, a recursive process is performed by Dividing-Cubes algorithm: each cube is classified as being inside, outside, or intersecting the surface; then, each cube intersecting the surface is divided to subcubes until the size of one projected cube is at most equal to the pixel size on the raster display (see Figure 3.4).

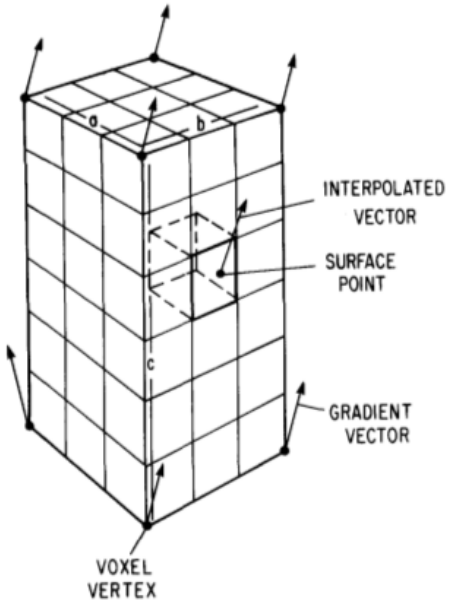


Figure 3.4: Subdivision of a cube (voxel) into smaller cubes using the dividing cubes algorithm; the cube was divided into 3x3x6 cubes [CLL+88].

After this recursive process, Dividing-Cubes start projecting generated pixel-sized cubes that intersect the surface onto the viewing plane. The projecting process involves computing the gradient vector at the voxel centre point by interpolating the gradients on its eight vertices in order to display it according to the Phong shading model (see Figure 3.4).

Thus, the idea is to approximate the surface by a cloud of points instead of a mesh of triangles. This means that the polygonisation stage of the Marching-Cubes is no longer necessary.

As presented in sampling stage, Dividing-Cubes is different from MC essentially in this stage. Dividing-Cubes recursively divides the cubes which intersect the implicit surface until the cube size is not larger than the pixel size. Then, the active cubes will be rendered directly onto screen without polygonizing the surface inside cubes basing on the look-up table. There is no need for setting topological configurations for cubes, neither applying numerical interpolation to find surface points between vertices. This explains why there is no concern about the shape ambiguities over cells. Morever, rendering points instead of triangles pays off in terms of computational cost.

Algorithms based on ray-tracing or on direct projection of voxels (3D primitives) give high quality images, with relative transparency or color of some materials being taken into account [BMN96].

3.3.1.3 Marching-Tetrahedra algorithm

The Marching-Tetrahedra (MT) algorithm was first suggested by Shirley and Tuchman [ST90] for computing a triangular mesh that approximates an isosurface. It is another

attempt to solve the ambiguities of the Marching-Cubes. This technique does not suffer from the ambiguities in the Marching-Cubes algorithm.

MT algorithm is closely related to Marching-Cubes algorithm except in that MT decomposes the space into tetrahedra cells instead of cubic cells like Marching-Cubes.

MT is essentially the Marching-Cubes algorithm with the tetrahedral decomposition step for each cube [GVJ+09].

In MT, each cube is split into six irregular tetrahedra by cutting the cube in half three times, cutting diagonally through each of the three pairs of opposing faces. In this way, the tetrahedra all share one of the main diagonals of the cube. Instead of the twelve edges of the cube, there are nineteen edges: the original twelve, six face diagonals. and the main diagonal. Just like in Marching-Cubes, the intersections of these edges with the isosurface are approximated by linearly interpolating the values at the grid of points [Wik16]. Note that the tetrahedron edges align with those on adjacent box cells, there is a method for splitting the box into 5 tetrahedrons which doesn't have this property [Bou97].

The surface within each tetrahedron is approximated by two triangles at maximum [GVJ+09]. The size of look-up table is much smaller than since only four rather than eight separate vertices are involved per tetrahedron. There are six or five tetrahedra to process instead of one single cube. The process is unambiguous, so no additional ambiguity handling is necessary [Wik16]. Eight different topological configurations for a tertahedra as illustrated in Figure 3.5.

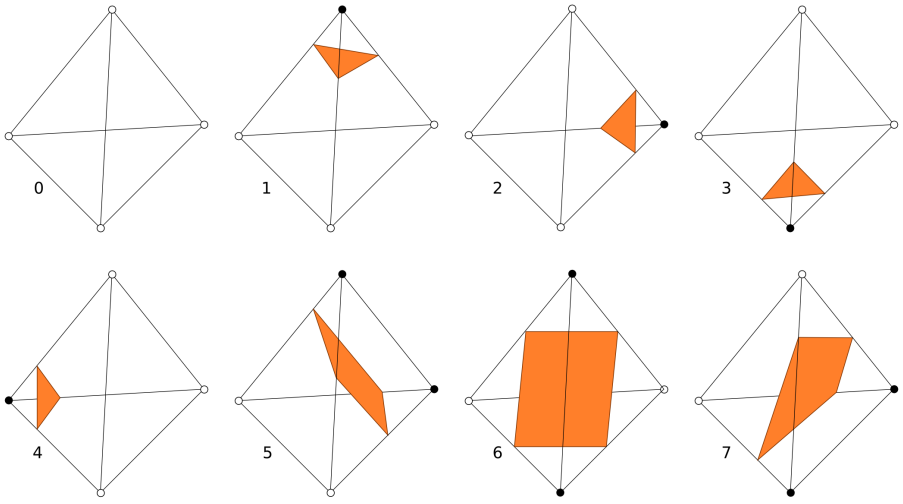


Figure 3.5: Marching tetrahedra topological configurations.

Each tetrahedron has sixteen possible configurations, falling into three classes: no intersection, intersection in one triangle and intersection in two (adjacent) triangles. It is straightforward to enumerate all sixteen configurations and map them to vertex index lists defining the appropriate triangle strips [Wik16]. Figure 3.6 illustrates the tetrahedral decomposition of a MC case, and the triangulation of the surface using this composition scheme.

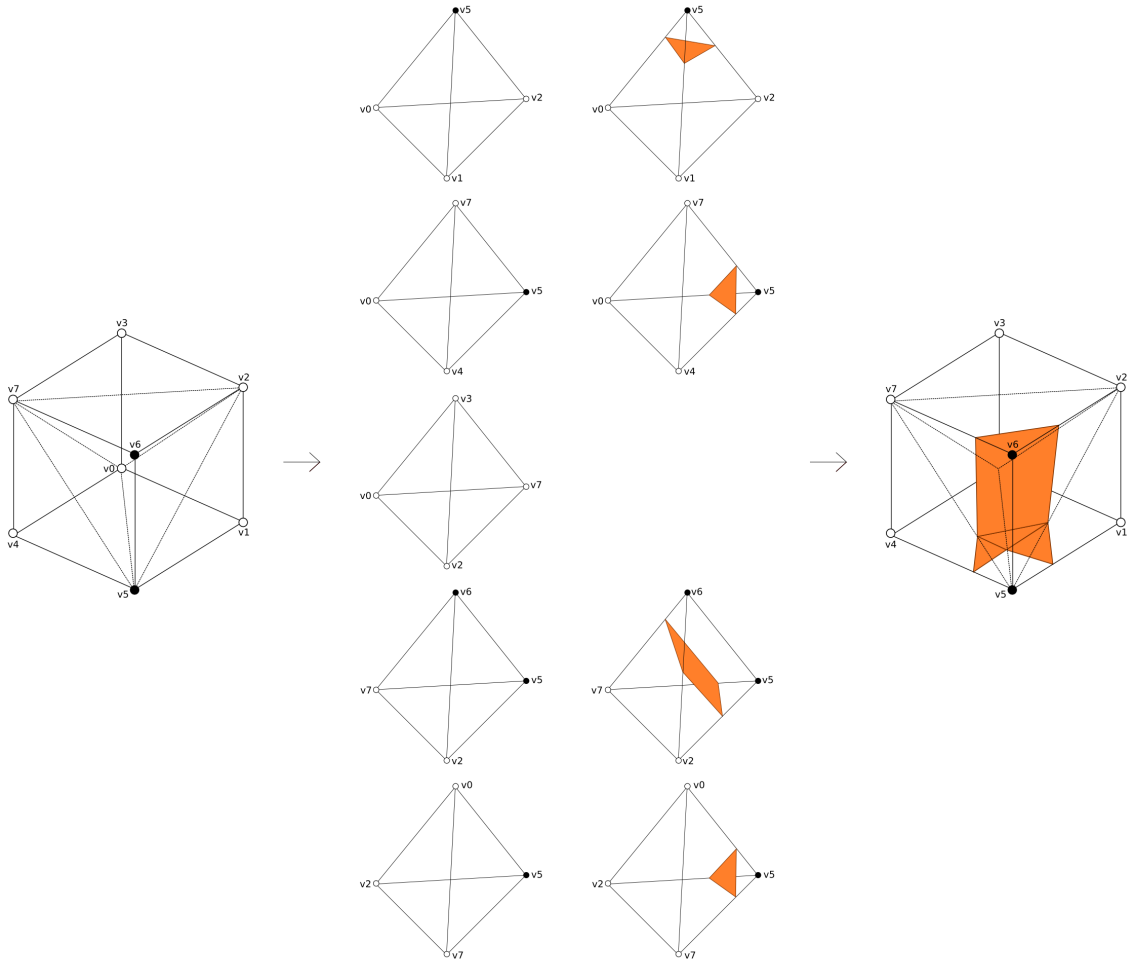


Figure 3.6: Marching tetrahedra decomposition and triangulation [ES].

Using tedtrahedra has the following advantages [GVJ+09]:

- Generality: It works on both unstructured and structured meshes. This makes the Marching-Tetrahedra a generic solution for isosurface extraction on all grid types.
- Disambiguation: The second advantage is that tetrahedra are less prone to shape ambiguities. The main reason behind this is that the number of configurations inside a cube.

The tetrahedra decomposition of the cube ends up with a set of tetrahedra within which the isosurface is correctly drawn as a plane. Note that we are here assuming that a linear model is being used. However, if the data vary trilinearly within the cubic cell, as is the case in the Marching-Cubes, then such a tetrahedral decomposition may not be free of ambiguities. Therefore, it is not correct to assume linear variation of data along the edges of a tetrahedron. That is, no claim can be made about the automatic removal of ambiguities of Marching-Cubes by simply decomposing cubes into tetrahedra [GVJ+09].

The downside is that the tessellation of a cube with tetrahedra requires a choice regarding the orientation of the tetrahedra, which may produce artificial ”bump” in the

isosurface because of interpolation along the face diagonals [Wik16].

3.3.2 Spatial continuation

Spatial continuation is an approach that combined exhaustive enumeration and continuation methods [GVJ+09, WMW86, BW97]. Spatial continuation method starts by decomposing the space into align-axis cells (i.e., cube), after that, the continuation scheme is applied to determine which cells intersect the surface before triangulating the surface in those cells.

In the continuation process, one cell which intersects the surface is identified firstly, and it is used as a seed cell. From this seed cell, the surface is tracked through adjoining cells, using shared edges with surface intersection. This process continues until the entire surface is enclosed by the collection of cubes. The surface contained in the cubes is then triangulated using look-up table. Figure 3.7 shows the implicit surface triangulated using Spatial continuation method.

Wyvill et al. [WMW86] were the first that proposed this approach, together with suggestions for improving the performance such as using the hash table to store only cells that intersect the surface. However, this method also produces cracks in the surface because of the ambiguities as described above.

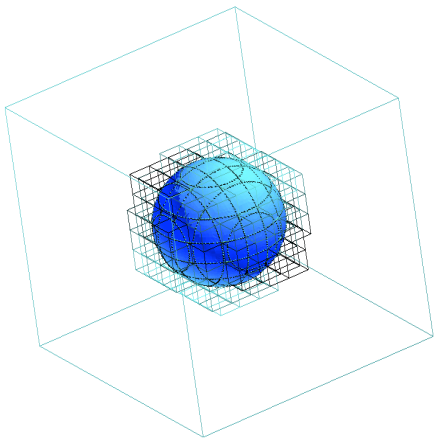


Figure 3.7: The triangulated implicit surface $f(x, y, z) = (x^2 + y^2 + z^2 - 2)^2$ triangulated using spatial continuation method [GDM10].

3.3.3 Spatial subdivision

3.3.3.1 Octree Subdivision

Octree is a tree data structure in which each internal node has exactly eight children. It is a 3D analogue of quadtrees. Octrees are most often used to partition a three dimensional space by recursively subdividing it into eight octants (or boxes) by three

hyperplanes (usually, axis-aligned planes) [GVJ+09, Mea80, Mea82, FA85, Wik15]. Octrees store infomation about curves, surfaces or volumes in a 3D space [GVJ+09].

The use of octrees for 3D computer graphics was pioneered by Donald Meagher [Wik15] as described in [Mea80]. Since an octree is just a collection of different size cubes, many algorithms have been developed using octrees to increase the efficiency of the sampling and triangulation process.

An octree is constructed by recursively subdividing space into eight cells (see Figure 3.8) until the remaining number of objects in each cell is below a pre-defined threshold, or a maximum tree depth is reached [Gel14].

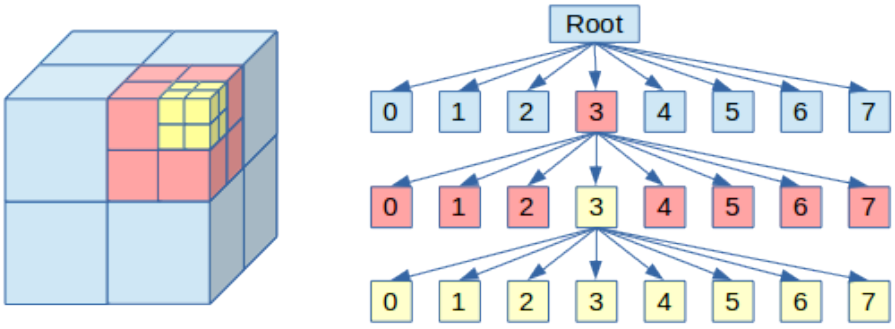


Figure 3.8: General structure of an octree [Gel14].

The idea behind the octree subdivision is to provide an adaptive approximation to implicit surfaces. That is, a cubic box through which the surface passes is subdivided into eight smaller boxes. These smaller cubes are stored into an octree data structure [GVJ+09].

The main advantages of spatial subdivision using octree represntations are that they are easy to generate and can achive any desired accuracy simply by decreasing the cell size. Many algorithms use octrees to store data of the space to tracking the surface [Gol09]. However, the main disadvantage of octree representation is that they can grow large with highly accurate surfaces, then they can be extremely inefficient [Gol09, GVJ+09].

In generaly, their importance come from the fact that they reinforce the adaptivity of the approximation to the implicit surface. However, these adaptive criteria are not sufficient to resolve all the topological ambiguities [GVJ+09].

3.3.3.2 Tetrahedral Subdivision

Although Marching-Tetrahedra approaches can overcome ambiguity, the different tetrahedral subdivisions of the cube can produce different facetizations. Specifically, there are two schemes to subdivide a cube into five tetrahedral-shaped cells [NY06].

Tetrahedral subdivision was used to achieve an adaptive mesh by Hall and Warren [HW90]. Algebraic implicit surface are polygonized using a recursive tetrahedron-based adaptive spatial subdivision method. This approach was extended by Hui and Jiang Hui

and Jiang to present an adaptive marching tetrahedral algorithm for bounded implicit patches. The patch is initially enclosed by a tertahedron that is subdivided according to vertex value, which results in an adaptive polygonization. Crespin also proposes an algorithm based on tetrahedral. This method has the limitation of generating a dynamic triangulation for VIS using incremental Delaunay tetrahedralization. An extended bounding box is created using the constraint points of the implicit model, which is subdivided into tetrahedra instead of cubical cells. A refinement creterion based on the tetrahedron's circumscribing sphere is used to subdivide the tetrahedron, being triangulated in a method similar to Bloomenthal's approach.

The first adaptive tetrahedral subdivision scheme to polygonise implicit surfaces was proposed by Hall and Warren [HW90]. Hall-Warren's algorithm performs an adaptive partition of space into tetrahedra. Interestingly, and regardless of the subdivision level, this tetrahedral subdivision enjoys the honeycomb property, i.e. the collection of all tetrahedra forms a honeycomb, the 3D analogue of a tessellation.

The algorithm may start with either a regular tetrahedra or with a tetrahedral mesh on the domain, though a nonregular tetrahedron could be also used by treating it as if it were regular. The nonregularity of a tetrahedron does not break the honeycomb property of the subdivision scheme because the subdivision of an irregular original tetrahedron is just the image under a linear transformation of the subdivision of a regular tetrahedron [HW90].

The Hall-Warren algorithm consists of five major stages [GVJ+09]:

- Decomposition of the bounding box into cubes.
- 5-tetrahedral decomposition of cubes.
- Uniform subdivision of tetrahedra.
- Adaptive subdivision of tetrahedra.
- Polygonisation of tranverse tetrahedra.

The first stage decomposes an axially aligned bounding box into a grid of cubes. At the second stage, these cubes are partitioned by using, for example, a 5-tetrahedral decomposition. The third stage performs a uniform subdivision of the tetrahedra down to a given minimum level l_MIN, regardless of whether the surface crosses a tetrahedron or not. Therefore, l_MIN works as a stopping condition for the first stage of the algorithm. The fourth stage concerns the adaptive subdivision of tetrahedron depends on the curvature of the surface. Finally, the algorithm performs the polygonisation of the surface inside each transverse tetrahedron [GVJ+09].

3.4 Healing algorithms

So far in this chapter, the reviewed algorithms are concerned with manifold implicit surfaces. Surfaces defined implicitly in 3D geometric modelling are limited to 2-manifolds because the corresponding implicit fields are usually defined by real-valued functions that bisect space into interior and exterior [GVJ+09, YKI02]. For instance, the most famous being the Marching-Cubes algorithm, it enumerates all cases of surface intersections with the cube composed of eight adjacent voxels, and then produces triangulated isosurface patches by means of a look-up table and linear interpolation. The resulting polygonal surfaces are 2-manifold [KTU+05].

Respect to the non-manifold surfaces, there are not many proposed solutions to solve its typical problems such as singularities or self-intersection. This can be seen in recent survey of implicit surface polygonzation of De-Araújo et al. [DALJ+15].

The beginning asumption for most of the surface reconstruction algorithm is that implicit function is a smooth function and the surface has no singularities.

Non-manifold features of implicit surfaces arise a series of problems to polygonisers. The main problem comes from the fact that the dimension may not be homogeneous. Dimension is a topological invariant, so if the dimension of a surface is not uniform, the polygoniser will face serious difficulties in keeping topological guarantees. The integration of a singularity solver into a polygoniser remains an open issue in computer graphics, regardless of whether the nature of the polygoniser, either continuation-based polygoniser or space partitioning-based polygoniser [GVJ+09].

For non-manifold implicit surfaces, there are several significant polygonizers proposed by Bloomenthal et al. [BF95], Gomes et al. [GDM10], Yamazaki et al. [YKI02]. Followings are an overview of those non-manifold polygonizers for implicit surfaces.

Bloomenthal and Ferguson [BF95] proposed a strategy for polygonizing nonmanifold implicit surface that used a new tetrahedron classification, as shown in Figure 3.9. The algorithm can handle with boundaries and intersections, but the existence of multiple regions significantly adds to the complexity of the polygonizer [WOK05]. The model of implicit surfaces in their work differs from the usual one in that a 'multiple regionalization' is employed basing on the observation that with the non-manifold polygonizer, a face must contain a region pair of interest.

Their algorithm utilize the cube as the propagating cell and decompose it into tetrahedra. Tetrahedra interseting the surface are polygonized concurrently with cube propagation. Surface vertices propagation directions are controlled by surface vertices. Surface vertices here are the intersection points between the edges of a tetrahedron face with the surface.

Yamazaki et al. [YKI02] also proposed an approach for nonmanifold implicit surface by extending the Marching-Cubes algorithm to correctly handle discontinuous fields (see Figure 3.10). Features such as holes, boundaries, and intersections are dealt with by en-

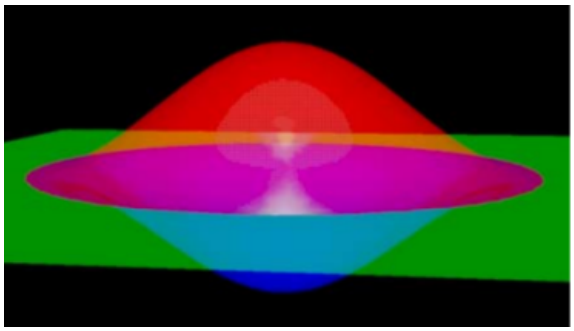


Figure 3.9: Non-manifold implicit surface by Bloomenthal et al. [BF95].

hancing the distance field, using bounding volumes to simplify the calculation between points. Although features are correctly approximated, the mesh quality depends on a user-defined subdivision level. If the cell size is small enough, then the triangulation is good and identifies sharp features. Otherwise, the mesh is of poor quality [DALJ+15].

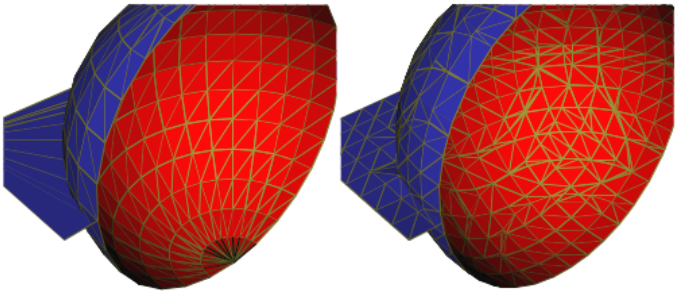


Figure 3.10: Non-manifold implicit surface by Yamazaki et al. [YKI02].

Yamazaki et al. treated a discontinuous distance field and prevented undesirable interpolation by generating the interpolation process by introducing the vertex generation diagram (VGD) which enables not only boundary representation but also the non-binary classification necessary for representing branches of faces [KTU+05].

To overcome limitations of most algorithms found in the literature and built-in commercial software packages, Gomes et al. [GDM10] proposed a general algorithm to polygonize nonhomogeneous and self-intersecting implicit surface that preserve local and global topological shape. By introducing a new uniform space space partitioning-based algorithm, their algorithm is capable of rendering surfaces with self-intersections and isolated 0 and 1D singularities (see Figure 3.11). In addition, their algorithm can correctly render isolated points, self-intersections, cut points, and isolated and dangling 1D surface patches [DALJ+15].

All three mentioned algorithms above belongs to Spatial partitioning class. With this class of algorithm, the traditional level set method usually ultilizes the signed-distance function f to represent the interface; also, the sign of f is responsible for indicating different spatial regions. Since the sign of a real value has only two choices (positive and negative), the signed-distance function f is only capable of binary regionalization which limits the traditional level set method within two-manifold interfaces. Some researchers [BF95, YKI02] in the geometry modeling field have proposed a new idea

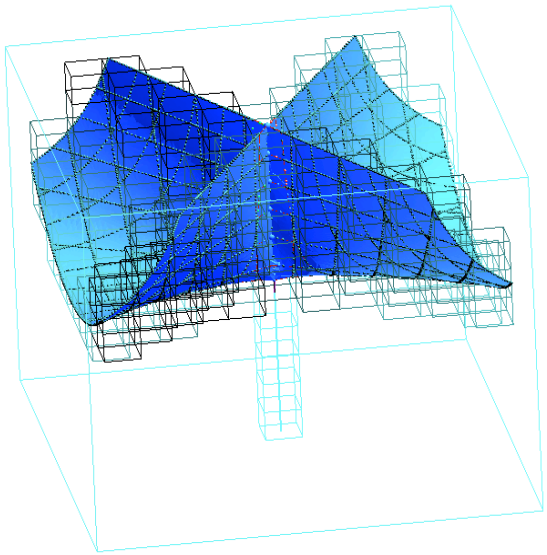


Figure 3.11: Non-homogeneous non-manifold implicit surface by Gomes et al. [GDM10].

to break the limit of the signed-distance function. Instead of using the sign of a real-valued function for binary regionalization, they use a region code to allow multiple regionalization. The region code is an integer value which has umlimited number of choices; thus, it can indicate as many regions as needed [ZYP09].

The problem of triangulating implicit surfaces has been investigated by Bajaj, Bloomenthal, Tristano, Owen and Canann, Lau and Lo, and Cuilliere. The marching cube algorithm of Lorensen and Cline can be used to triangulate an implicit surface. The algorithm determines the edges of a cubic grid intersecting the surface and then generates a tesselation by connecting these intersection points. Although the algorithm is very simple, there is no guarantee that the output has the same topology as that of the surface. Stander and Hart proposed varying the value of the implicit function from negative infinite to positive infinite and dynamically maintaining a triangulation of the changing isosurface. It is necessary to track all critical points of the implicit function. Maintaining triangulations of isosurfaces is a huge overhead given that only one isosurface. At equilibrium, the particles can be connected to form the surface triangulation, but it unclear how to ensure that the surface topology is captured. The above algorithms do not offer any guarantee on the triangle shape though some of them include heuristics and illustrate their effectiveness experimentally [CDRR07].

3.5 Concluding Remarks

This chapter has reviewed the relevant literature concerning triangulation of implicit surfaces, with a particular focus on space partitioning algorithms and non-manifold implicit surfaces. In the next chapter, we will introduce our algorithm for triangulating such non-manifold implicit surfaces, which can be seen as an extension to marching

cubes (MCs) algorithm. It is called *healed marching cubes*, and makes use of differential geometry and numerical methods to resolve singularities locally inside each cube.

Chapter 4

Healed Marching Cubes Algorithm

No many algorithms have been developed for non-manifold surfaces so far, largely because of the difficulties in resolving singularities, either symbolically or numerically. In this dissertation, we manage to resolve such singularities using a numerical strategy. This numerical approach allows us to extend the traditional marching cubes triangulation algorithm to non-manifold implicit surfaces with self-intersections. Such an extended algorithm is here called *healed marching cubes* (HMC).

4.1 Introduction

As seen in the previous chapter, the marching cubes (MC) algorithm belongs to the category of spatial partitioning methods. More specifically, each surface is mapped into an axis-aligned cube grid in order to sample the surface against the edges of all cubes.

It happens that the MC algorithm does not produce a topologically non-ambiguous triangulation for non-manifold implicit surfaces. This is so because the triangulation of the surface in the vicinity of singularities is not performed correctly. In order to fix this problem, we have designed an extension to MCs, called *healed marching cubes* algorithm. This algorithm is capable of re-triangulating the surface inside each cube containing any singularity, i.e., it is capable of re-triangulating the wrong pieces of the surface in an automated manner.

4.2 Ambiguous Topological Configurations

The topological configurations of MCs only apply to manifold surfaces. Consequently, when it comes the time of non-manifold surfaces, some of the topological configurations become ambiguous. We have identified three ambiguous topological configurations in the reduced set of 15 configurations, but this number of ambiguous configurations scales in the enlarged set of 256 configurations.

Those three ambiguous configurations in the reduced set of configurations are shown in Figure 4.1), and as follows:

- 10th out of 15 reduced topological configurations; this configuration also represents the topological configurations numbered (in decimal) as 60, 85, 105, 150,

170, 195.

- 12th out of reduced 15 topological configurations, this configuration also represents the topological configurations numbered (in decimal) as 30, 45, 53, 58, 75, 83, 86, 89, 92, 101, 106, 120, 135, 149, 154, 163, 166, 169, 172, 180, 197, 202, 210, 225.
- 13th out of reduced 15 topological configurations, this configuration also represents the topological configurations numbered (in decimal) as 90, 165.

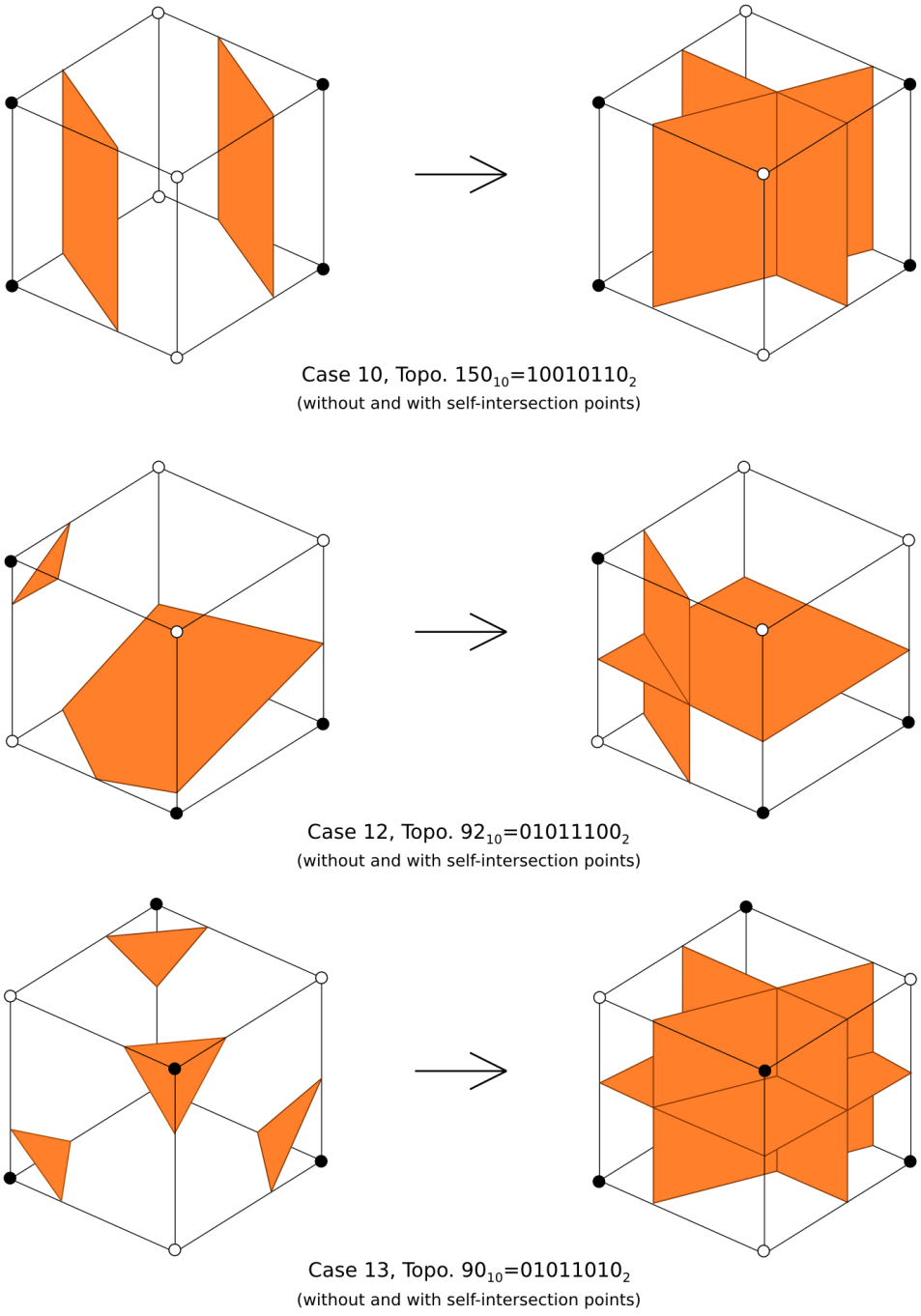


Figure 4.1: The three ambiguous topological configurations of MCs.

Note that each one of those three configurations or cases contains at least two components (i.e., two pieces of the surface within the cube), as shown on the left hand side in Figure 4.1); more specifically, the cases 10 and 12 have 2 components each, while the case 13 has 4 components. In the reduced set of 15 configurations of the standard MC algorithm, there are 7 configurations with at least two components. However, only 3 configurations are considered by HMC algorithm because at this point in time, HMC is interested in cubes possibly containing more than one self-intersection point. With 4 other remaining cases, they can contain at most one self-intersection point assuming that standard MC obtains an accurate result in the sampling stage.

Self-intersection points can lie on the edges, inside the faces, or inside the cube. For instance, cube of Case 13 in Figure 4.1) contains six self-intersection points inside the surface, and one self-intersection point inside the cube. Cubes with self-intersection on the edges will be mentioned in the next section.

4.3 HMC Algorithm

The healed marching cubes algorithm works into two stages. First, we compute the marching cubes triangulation as usual for manifold surfaces. Second, we proceed to the healing process in case we are in the presence of non-manifold surfaces, i.e., surfaces with self-intersections.

The question is how to find and delineate lines that result from the self-intersections of a given analytic surface, in order to fix its topology. In general, after completing the first stage concerning the marching cubes algorithm for manifold surfaces, the healed marching cubes algorithm enters in action as follows:

1. Determine cubes with self-intersection points
2. Determine self-intersection points in each cube
3. Add the correct triangles and remove the wrong ones
4. Rendering the triangles

The following subsections describe these four steps in more detail. This description will be accompanied by the triangulation of the implicit surface that results from the union of a plane and a sphere, which is analytically given the function $f(x, y, z) = x^2\,z + y^2\,z + z^3 - z$, and illustrated in Figure 4.2.

4.3.1 Determining critical cubes

At this point, it is assumed that the marching cubes algorithm has already been executed (see Figure 4.2(a)), so that it is time to heal the cubes with self-intersections, also

called critical cubes. First, the geometric healing algorithm finds cubes that intersect the surface (see Figure 4.2(b)). These critical cubes are then saved into a 1-dimensional array or vector in C++; more specifically, their indexes (i, j, k) in the grid of cubes are saved.

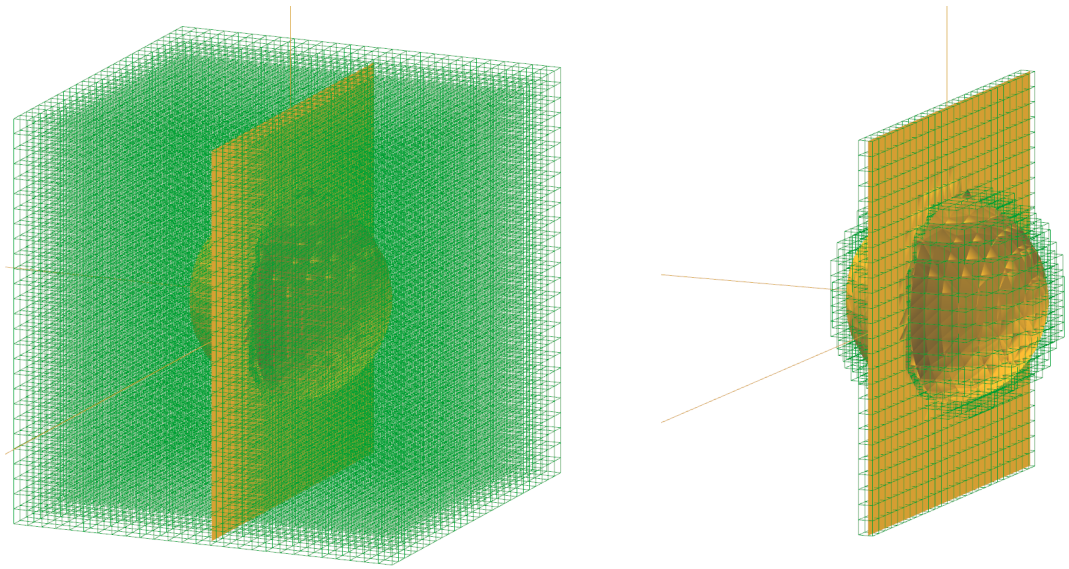


Figure 4.2: The implicit surface defined by the real function $f(x, y, z) = x^2 + y^2 + z^3 - z$: (a) after the discretization of the rectangular domain into cubes (in green) of length 0.15; (b) cubes (in green) intersecting the surface.

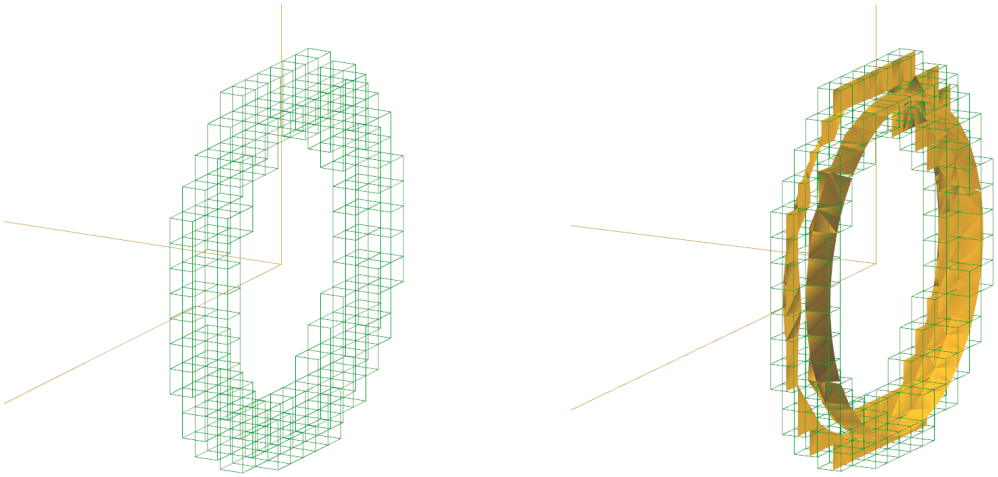


Figure 4.3: Critical cubes: (a) critical cubes with ε=0.2; (b) critical cubes with wrong pieces of the surface.

Second, the algorithm filters out the cubes where they likely possess self-intersections (e.g., points and lines of self-intersection) of the surface (see Figure 4.3). To determine which cubes on the surface contain self-intersection points (i.e., critical cubes), we take advantage of the following critical point test condition:

$$\left|\frac{df}{dx}\right| + \left|\frac{df}{dy}\right| + \left|\frac{df}{dz}\right| < \varepsilon \tag{4.1}$$

where ε stands for a small tolerance. This is so because the partial derivatives vanish at any critical point simultaneously.

In order to decide whether a cube is critical or not, HMC calculates the partial derivatives on the vertices of the triangles inside the cube checking whether there is one that satisfies the condition 4.1. If no vertex satisfies such condition, that means that the cube is not critical.

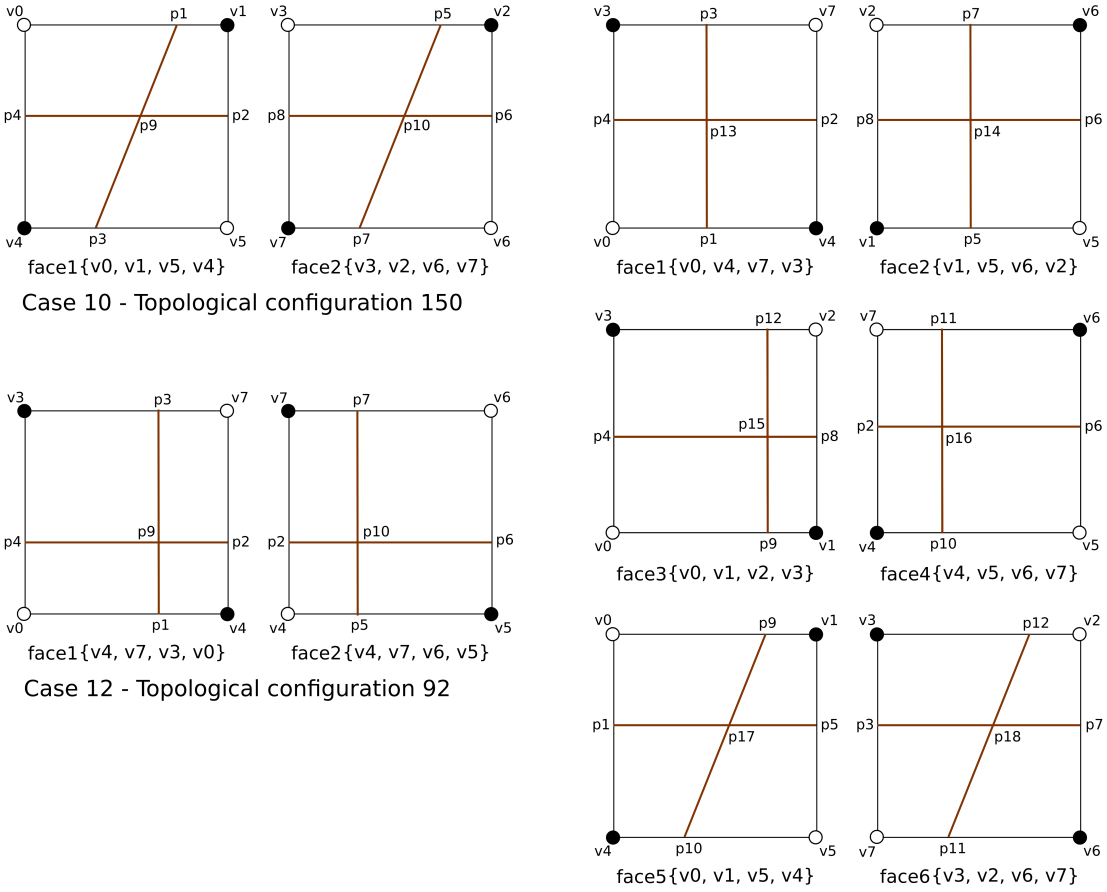
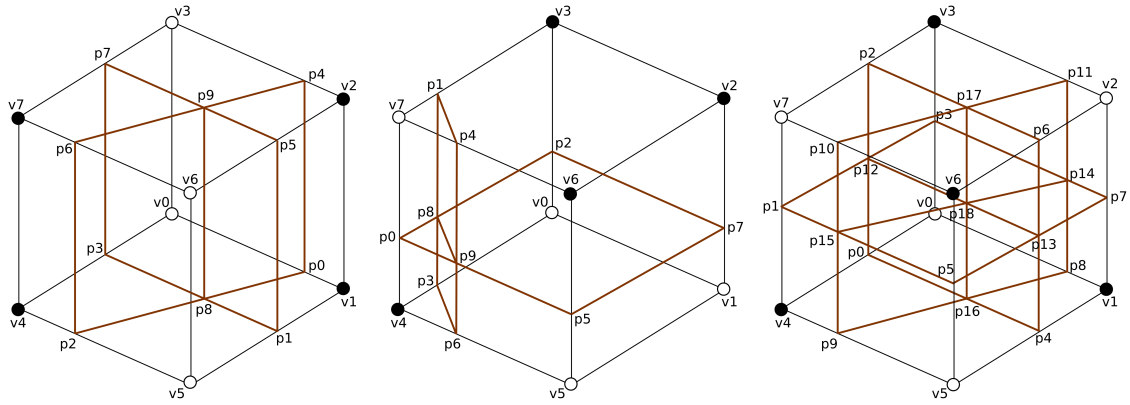


Figure 4.4: An illustration of critical cube faces formed by the arrangement of cube's vertices of different topologies of HMC cases that contain self-intersection points

4.3.2 Determining self-intersection points

In this step, HMC determines all self-intersection points existing in each critical cube.

As can be seen from HMC cases, self-intersection points lie on the cube faces where there are two line-segments intersect each other. In other words, self-intersection points lie on cube faces where all their four edges intersect the surface at a point



Case 10, Topo. 150_{10}=10010110_2 Case 12, Topo. 92_{10}=01011100_2 Case 13, Topo. 90_{10}=01011010_2

Saving order of surface-edge intersections and self-intersection points

p0	p1	p2	p3	p4	p5	p6	p7	p8	p9

p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18

Figure 4.5: Fixed position of points in critical cubes of HMC cases and the saving order of those points in a vector: (a) position of points of critical cubes of Case 10; (b) position of points of critical cubes of Case 12; (c) position of points of critical cubes of Case 13; (d) the saving order of surface-edge intersections and self-intersection points of critical cubes of Case 10 and 12 in a vector; (e) the saving order of surface-edge intersections and self-intersection points of critical cubes of Case 13 in a vector

along each edge. Then, basing on the symmetrical property of the cube, for different topologies of a HMC case, HMC uses the same procedure of determining self-intersection points by arranging the cube vertices to form the cube faces containing self-intersection points as input (see Figure 4.4). Then, the problem of determining self-intersection points is reduced to the finding intersection point between two line-segments on each face. This can be solved by using a root-finding method (e.g., secant method). The algorithm which HMC used to determine intersection point between two line-segments is described in the Appendixes.

With HMC case 13, self-intersection points can lie inside critical cube. HMC determines such points by applying the same root-finding method to find the intersection point between any two self-intersecting line-segments.

HMC saves the self-intersection points discovered together with surface-edge intersections of a critical cube in a 1-dimensional array or vector. These arrays or vectors are contained in another array or vector (i.e., 2-dimensional array or vector) which the size is equal to the size of array or vector used to save the critical cubes. This is have the meaning in determine a points set is belonged to which critical cubes.

Using the vertices and points numbering in Figure 4.5, the order of self-intersection points and surface-edge intersections saved in in a vector is as in Figure 4.5. The first saving order is for critical cubes of HMC case 10 and 12, the second saving order is for critical cubes of HMC case 13. Those fixed saving orders are leveraged later in the next step to triangulate the surface pieces in critical cubes.

Figure 4.6 shows the self-intersection points found in critical cubes with plane-and-ball surface with res=0.15.

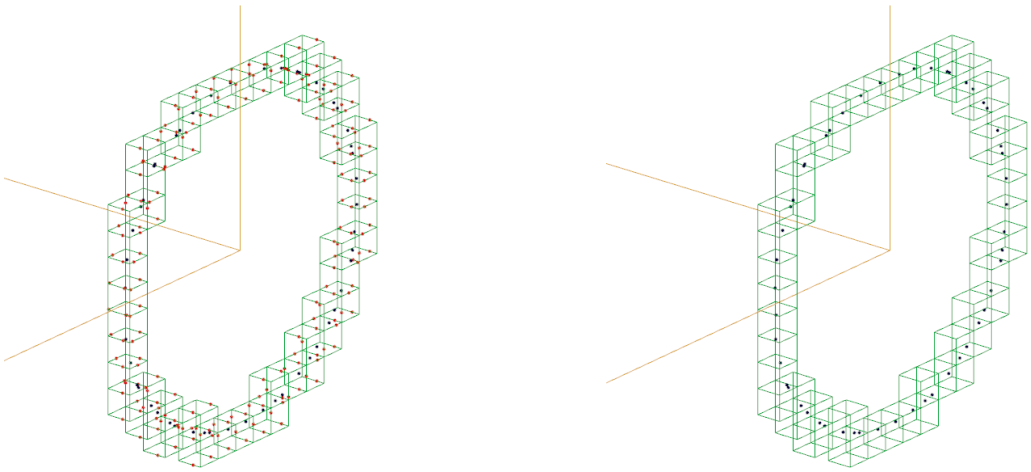


Figure 4.6: Self-intersection points determined with plane-and-ball surface with res=0.15: (a) surface-edge intersections (in red) and self-intersection points (in blue) (b) self-intersection points

4.3.3 Add the correct triangles and remove the wrong ones

After the third step, we've had all necessary points to triangulate the surface in critical cubes. The position of surface-edge intersections and self-intersection points saved in a vector are fixed, HMC starts triangulating by placing the vertices in the appropriate orders to form the triangles.

The self-intersection line-segments connect points of different planes of the surface within a critical cube. In all three HMC cases, there are three types of planar points (i.e., points belong to a certain plane) to triangulate: planes with three points, four points, and five points. Figure 4.7 shows the triangulations used in which self-intersection line-segment roles as the center line connecting the planar points, and the priority to connect the points to the self-intersection line-segment is based on the saving order of those points in the array or vector. The triangle vertices by HMC are saved in a 1-dimensional array or vector contained in another array or vector (i.e., 2-dimensional array or vector) whose size is equal to the array or vector of critical cubes.

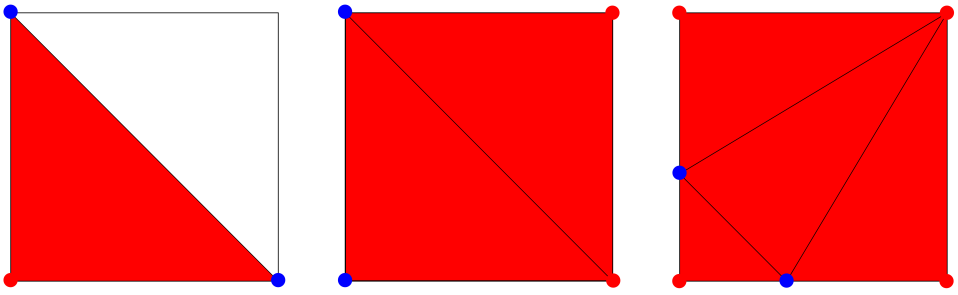
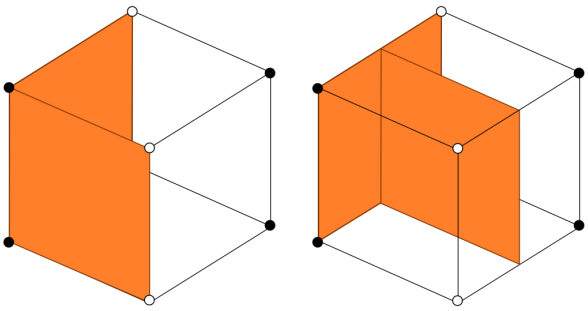
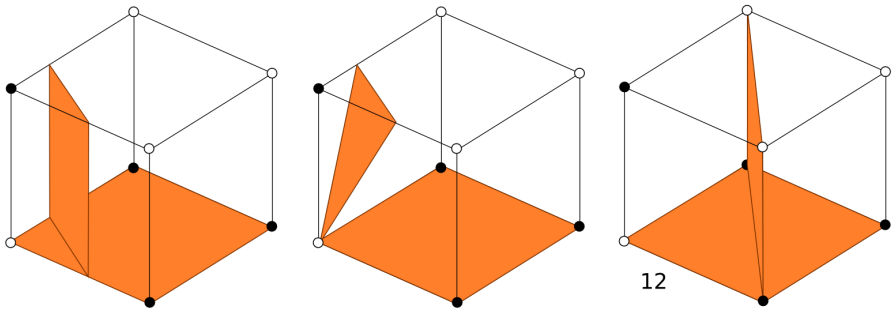


Figure 4.7: Triangulations for planar types of the surface within critical cubes.

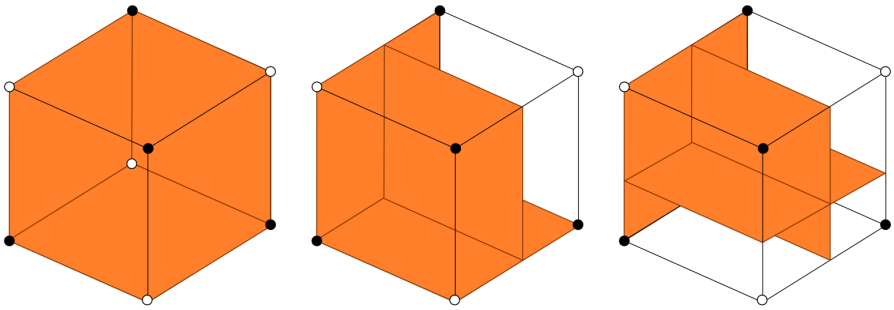
When the self-intersection points lie on the cube edges, they overlap the surface-edge intersections (see Figure 4.8). To avoid triangulating overlapped points, HMC checks the overlap of each triple of vertices before putting them in the vector of triangle vertices.



Case 10 - Topological configuration 150



Case 12 - Topological configuration 92



Case 13 - Topological configuration 90

Figure 4.8: Possible overlaps between self-intersection points with surface-edge intersections, and between self-intersection points themselves.

Figure 4.9 shows the triangles produced by healed marching cubes for the plane-and-ball surface with res=0.15. To remove wrong cubes, the triangle vertices associated to HMCs are combined with ones produced by MCs in which the healed cubes replace the cubes of marching cubes.

4.3.4 Rendering the triangles

The combined vector of marching cubes and healed marching cubes triangle vertices are used to render the non-manifold implicit surface. We calculate the gradient vector

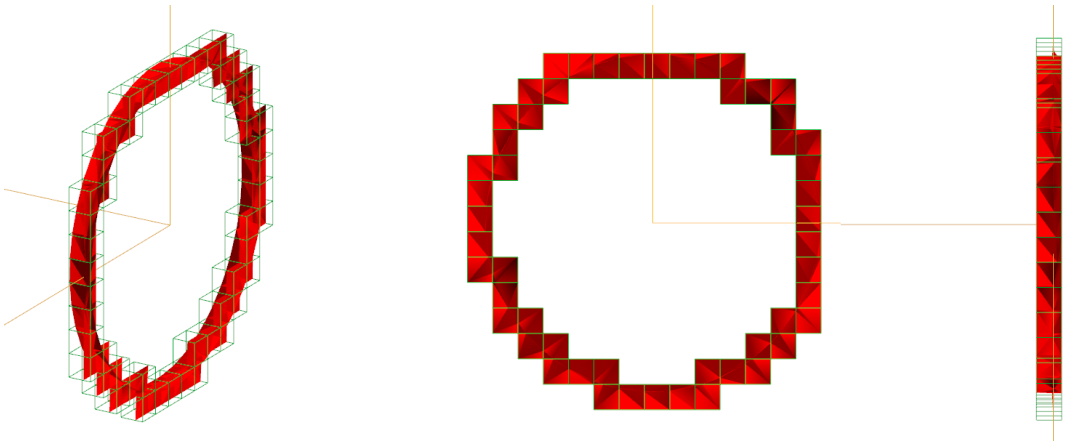


Figure 4.9: The result of triangulated surface within critical cubes of plane-and-ball with res=0.15 from different view points.

$(\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz})$ as the normal to each vertex. The rendering algorithms use these normal to produce Gouraud/Phong shaded-images.

4.4 Results

4.4.1 Hardware/software Setup

Having applied HMCs on different non-manifold implicit surfaces, obtained results are represented below. For surfaces and resolution that the standard MC samples the surface correctly, we get the results as shown in Figure 4.10, Figure 4.11, Figure 4.12, Figure 4.13, for each non-manifold implicit surface, we show the surface constructed by standard MCs on the left, and the healed surface by HMCs on the right, and the details of each rendered surface are also mentioned.

The detail configurations of the computer used to execute the program is as follows:

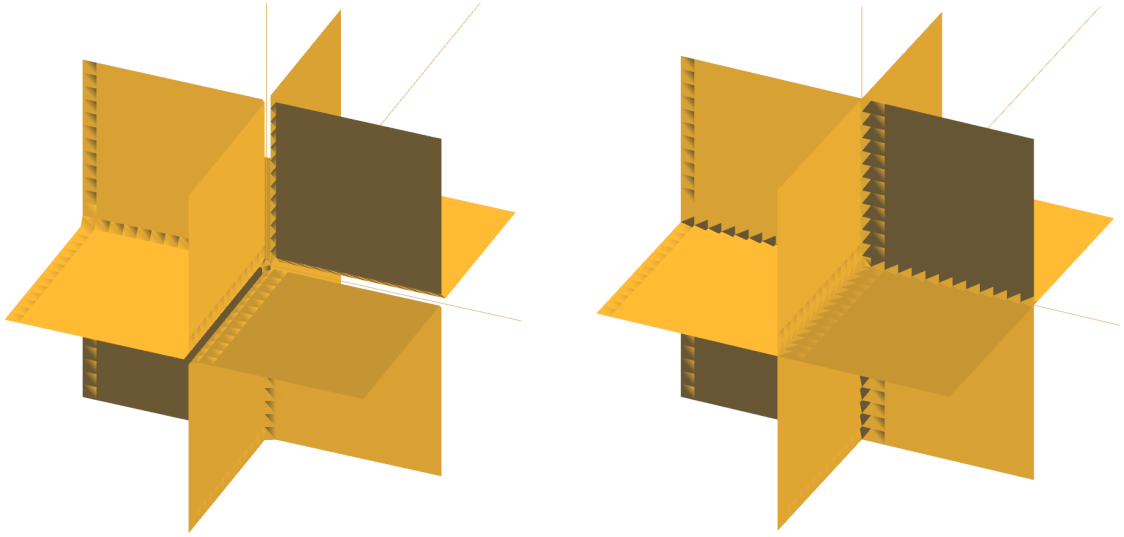
- Hardware setup:
  - CPU: Intel(R) Core(TM) i7-2720QM CPU @ 2.20GHz (4 cores enabled, 8 threads enabled)
  - RAM: 16GB
  - Graphics card: NVIDIA GF108GLM Quadro 1000M
- Software setup:
  - Operating system: Ubuntu Desktop 14.04 LTS
  - Graphics library: OpenGL version 4.2.0
  - Programming language: C++11 standard
  - Compiler: gcc version 4.8.4

4.4.2 Xano Surface

Xano surface is represented by implicit function:

$$f(x, y, z) = x\, y\, z \tag{4.2}$$

The representation of this surface is shown in Figure 4.10.



(a) (b)

Figure 4.10: Xano surface with res=0.15; (a) surface triangulated by MCs; (b) surface triangulated by HMCs.

Statistics of produced triangular mesh:

- Number of triangles by standard MC: 4054
- Number of triangles by HMC: 4374 (in which, correct triangles by MC are 3750, number of triangles used for healing the critical cubes is 624)

Statistics of running time until the end of triangulation step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	13.00	13.00	11.00	10.00	10.00	19.00	18.00	15.00	14.00	18.00
HMC	19.0	16.00	25.00	21.00	21.00	14.00	16.00	12.00	14.00	11.00

Statistics of running time until the end of rendering step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

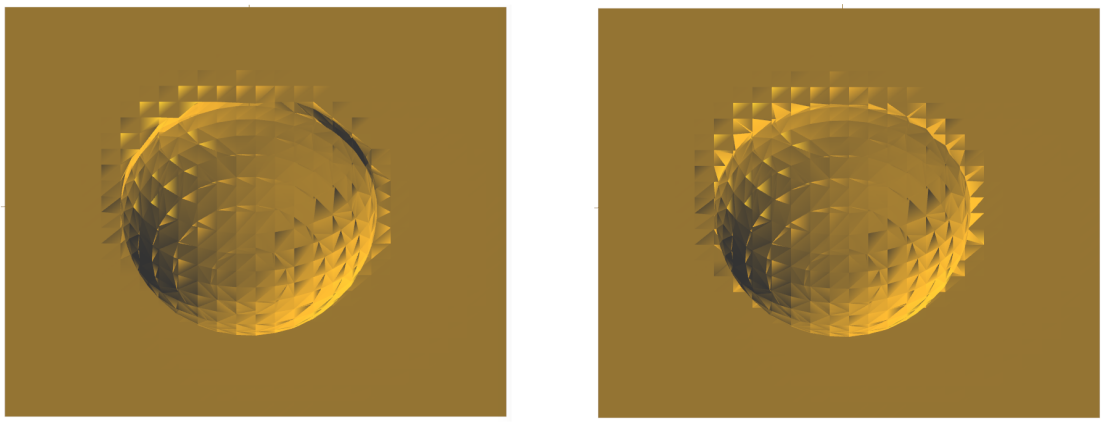
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	136.00	206.00	195.00	147.00	140.00	206.00	138.00	158.00	147.00	196.00
HMC	136.00	206.00	191.00	149.00	223.00	196.00	113.00	221.00	215.00	140.00

4.4.3 Plane-and-Ball Surface

Plane-and-Ball surface is represented by implicit function:

$$f(x, y, z) = x^2\, z + y^2\, z + z^3 - z \tag{4.3}$$

The representation of this surface is shown in Figure 4.11.



(a) (b)

Figure 4.11: Plane-and-ball surface triangulated with res=0.15; (a) surface triangulated by MC; (b) surface triangulatedby HMC

Statistics of produced triangular mesh:

- Number of triangles by standard MC: 3004
- Number of triangles by HMC: 3212 (in which, correct triangles by MC are 2796, number of triangles used for healing the critical cubes is 416)

Statistics of running time until the end of triangulation step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	11.00	7.00	8.00	8.00	13.00	9.00	14.00	14.00	17.00	7.00
HMC	14.00	23.00	22.00	10.00	19.00	14.00	21.00	15.00	9.00	18.00

Statistics of running time until the end of rendering step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

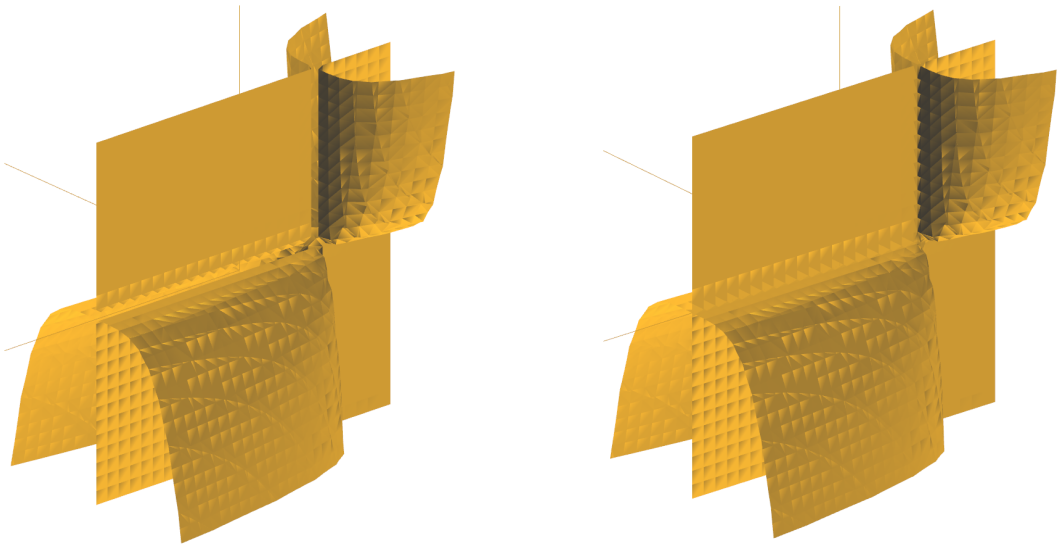
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	211.00	226.00	157.00	141.00	163.00	123.00	215.00	208.00	109.00	208.00
HMC	218.00	142.00	146.00	200.00	193.00	216.00	120.00	121.00	171.00	202.00

4.4.4 Zeppelin Surface

Diabolo surface is represented by implicit function:

$$f(x, y, z) = x\,y\,z + y\,z + 2\,z^5 \tag{4.4}$$

The representation of this surface is shown in Figure 4.12.



(a) (b)

Figure 4.12: Zeppelin surface triangulated with res=0.15; (a) surface triangulated by MCs; (b) surface triangulated by HMCs.

Statistics of produced triangular mesh:

- Number of triangles by standard MC: 4044
- Number of triangles by HMC: 4264 (in which, correct triangles by MC are 3840, number of triangles used for healing the critical cubes is 424)

Statistics of running time until the end of triangulation step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	14.00	14.00	21.00	17.00	10.00	11.00	8.00	8.00	14.00	20.00
HMC	16.00	16.00	26.00	15.00	21.00	14.00	16.00	21.00	18.00	11.00

Statistics of running time until the end of rendering step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

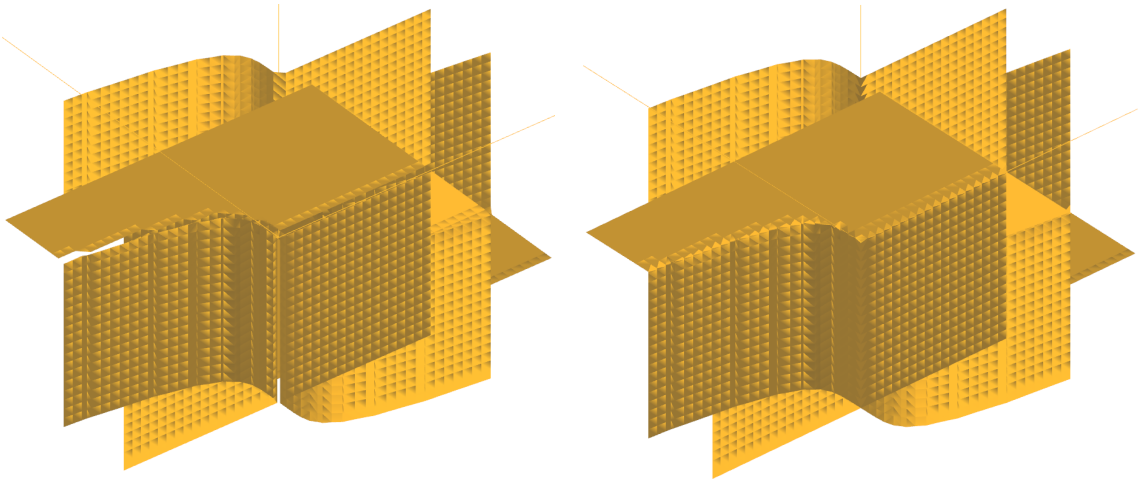
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	201.00	212.00	174.00	153.00	185.00	116.00	105.00	196.00	192.00	118.00
HMC	135.00	205.00	127.00	195.00	225.00	108.00	177.00	134.00	216.00	128.00

4.4.5 Diabolo Surface

Diabolo surface is represented by implicit function:

$$f(x, y, z) = x\,y\,z + 2\,y\,z^6 \tag{4.5}$$

The representation of this surface is shown in Figure 4.13.



(a) (b)

Figure 4.13: Diabolo surface triangulated with res=0.11; (a) surface triangulated by MCs; (b) surface triangulated by HMCs.

Statistics of produced triangular mesh:

- Number of triangles by standard MC: 9070
- Number of triangles by HMC: 9582 (in which, correct triangles by MC are 8574, number of triangles used for healing the critical cubes is 1008)

Statistics of running time until the end of triangulation step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	24.00	29.00	33.00	24.00	37.00	39.00	34.00	40.00	26.0	32.00
HMC	44.00	42.00	47.00	49.00	35.00	36.00	41.00	35.00	36.00	41.00

Statistics of running time until the end of rendering step (the measurement unit is milliseconds, and the real values are rounded to the integer part):

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
MC	170.00	183.00	136.00	210.00	217.00	224.00	253.00	242.00	136.00	146.00
HMC	173.00	223.00	170.00	147.00	160.00	207.00	175.00	237.00	254.00	229.00

4.5 Concluding Remarks

This chapter presented the details of HMC algorithm. The obtained results have shown that the HMCs works well basing on standard MCs in terms of accuracy and efficiency. The difference in running time is relatively small. Generally, HMCs can eliminate the incorrect pieces of surface within critical cubes and reconstruct the correct one to heal the gaps at self-intersection places on the surface.

Chapter 5

Conclusions

This thesis elaborates on the research work carried out in triangulating non-manifold implicit surfaces, which is an important research topic in geometric computing and computer graphics. In short, we can say that the main contribution of this work remains in the re-triangulation of critical cubes, i.e., those cubes inside which we find singularities of the implicit surface. Consequently, we end up extending the well-known marching cubes algorithm to non-manifold implicit surfaces. Such extended algorithm is here called *healed marching cubes*, or HMC for sort.

5.1 Research Context

First of all, let us refer that the work carried out in this thesis started after getting a better understanding of how difficult is to triangulate non-manifold implicit surfaces. This challenge comes from the fact there is not any triangulation algorithm in the literature capable of dealing with any non-manifold implicit surface. In other words, there is no such a *general* triangulation algorithm, so that this issue remains unsolved in geometric modeling and computer graphics.

5.2 Research Questions

Given the work performed during the current academic year on non-manifold triangulations, we are able now to respond to the following research questions:

Is it possible to triangulate non-manifold implicit surfaces?

In Chapter 4, we show that the answer to this question is affirmative. It is clear that these special surfaces are not directly triangulable. Essentially, we are using the principle behind the resolution of singularities as in [Shi97]. That is, we first find the singularities using differential geometry tools, and then each outstanding manifold component of the surface is triangulated as usual for manifold surfaces.

Is it possible to resolve singularities using numerical methods?

As shown in Chapter 4, we can use classical numerical methods to find singularities of non-manifold implicit surfaces. But, instead of finding zeros of the function that describes a given surface, we have to find zeros of its gradient (i.e., partial derivatives).

Summing up, and recalling the problem statement mentioned in Chapter 1:

> *Is it feasible to reformulate the marching cubes algorithm in order to cope with singularities?*

we able to say that the research work described in this thesis responds positively to such problem statement. Furthermore, the marching cubes revealed themselves adequate to triangulate non-manifold surfaces because the re-triangulation is confined to each critical cube. This locality of the re-triangulation procedure is not possible with other sorts of triangulation algorithms. For example, using re-triangulation together with mesh growing-based algorithms would be rather difficult to achieve because there is control on the locality of singularities.

5.3 Algorithm Limitations

During the study and design of the healed marching cubes algorithm (HMC), a few limitations were identified, which end up opening a window for future work, namely:

- *Sensitivity to grid resolution.* The preservation of the topological type of the surface depends on the grid resolution (i.e., voxel length) used in the marching cubes algorithm. It is clear that we can always use a very small voxel length, but this incurs in a very time-consuming task. The question then is which is the voxel length that results from the most adequate tradeoff between topological accuracy a computation efficiency.
- *Sensitivity to numerical sampling.* As in the traditional marching cubes, HMC algorithm depends on the accuracy of the numerical method used in sampling the implicit surface. But, more importantly, it depends on the numerical method employed in sampling singularities of implicit surfaces. Any missing singularity on the boundary or interior of a cube certainly results in inaccuracies at the topological level.

For brevity, let us also mention that we have not considered other types of singularities like point singularities inside a cube, or even point singularities on the boundary of a cube. But, these singularities are easily resolved using a numerical method that converges to zeros of the partial derivatives simultaneously.

5.4 Future Work

In the near future, the goal is to improve the algorithm in order to overcome its limitations mentioned above. In respect to grid resolution, we intend to use a hierarchical

scheme for the healed marching cubes, which recursively subdivides each critical cube to make sure about the computation of singularities and, consequently, to preserve the local topological type of the surface after its triangulation.

In regard to resolution of singularities in numerical terms, we intend to develop algorithms to march along 1-dimensional singularities directly, instead of finding them inside each cube. Finding point singularities (or 0-dimensional singularities) will follow a similar procedure. This procedure consists if finding the next minimum (in absolute value) in a small neighborhood of the current point; this is so because the first partial derivatives vanish simultaneously at a singular point.

Bibliography

[BCSV04] Jean-Daniel Boissonnat, David Cohen-Steiner, and Gert Vegter. Meshing implicit surfaces with certified topology. In *ACM STOC*, 2004. 1

[BF95] Jules Bloomenthal and Keith Ferguson. Polygonization of non-manifold implicit surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 309–316. ACM, 1995. xiii, 1, 26, 27

[BMN96] Fatima Boumghar, Serge Miguet, and Jean-Marc Nicod. Complexity of discrete surfaces in the dividing-cubes algorithm. In *Discrete Geometry for Computer Imagery*, pages 269–280. Springer, 1996. 20

[Bou97] Paul Bourke. Polygonising a Scalar Field using Tetrahedrons, 1997. Available from: http://paulbourke.net/geometry/polygonise/. 21

[BPK+07] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Röossl. Geometric modeling based on polygonal meshes. 2007. 8

[BS05] R.J. Balsys and K.G. Suffern. Adaptive polygonisation of non-manifold implicit surfaces. In *Proceedings of the 2nd International Conference on Computer Graphics, Imaging and Vision: New Trends* (CGIV'05), Beijing, China , July 25-29, pages 257–263. IEEE Press, 2005. 8

[BW97] Jules Bloomenthal and Brian Wyvill, editors. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. 1, 23

[CDRR07] Siu-Wing Cheng, Tamal K Dey, Edgar A Ramos, and Tathagata Ray. Sampling and meshing a surface with guaranteed topology and geometry. *SIAM journal on computing*, 37(4):1199–1227, 2007. 28

[Che95] Evgeni V Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. *Institute for High Energy Physics, Moscow, Russia, Report CN/95-17*, 42, 1995. 18, 19

[CLL+88] Harvey E Cline, William E Lorensen, Sigwalt Ludke, Carl R Crawford, and Bruce C Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. *Medical physics*, 15(3):320–327, 1988. 19, 20

[CX14] Andrey N Chernikov and Jing Xu. A Computer-Assisted Proof of Correctness of a Marching Cubes Algorithm. In *Proceedings of the 22nd International Meshing Roundtable*, pages 505–523. Springer, 2014. 16

[DALJ+15] B.R. De-Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. A Survey on Implicit Surface Polygonization. *ACM Computing Surveys*, 47(4):1–39, 2015. Available from: http://dl.acm.org/citation.cfm?doid=2775083.2732197. 14, 16, 26, 27

[Dür88] Martin J Dürst. Re: additional reference to marching cubes. *ACM SIGGRAPH Computer Graphics*, 22(5):243, 1988. 18

[ES] Carlos Hernández Esteban and Francis Schmitt. Silhouette and Stereo Fusion for 3D Object Modeling. Available from: http://carlos-hernandez.org/3dmodeling.html. xiii, 22

[FA85] Wm Randolph Franklin and Varol Akman. *Octree data structures and creation by stacking*. Springer, 1985. 24

[FO03] Remondino Fabio and Others. From point cloud to surface: the modeling and visualization problem. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34(5):W10, 2003. 13, 54

[GDM10] Abel Gomes, Sérgio Dias, and José Morgado. Polygonization of non-homogeneous non-manifold implicit surfaces with tentative topological guarantees. *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, pages 1–8, 2010. Available from: http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=5586012. xiii, 7, 14, 23, 26, 27, 28

[Gel14] David Geler. Advanced Octrees 1: Preliminaries, Insertion Strategies and Maximum Tree Depth, 2014. Available from: https://geidav.wordpress.com/2014/07/18/advanced-octrees-1-preliminaries-insertion-strategies-and-max-\protect\discretionary{\char\hyphenchar\font}{}{}tree-depth/. xiii, 24

[Gol09] Ronald Goldman. *An integrated introduction to computer graphics and geometric modeling*. CRC Press, 2009. 24

[GVJ+09] Abel J.P. Gomes, Irina Voiculescu, Jorge Joaquim, Brian Wyvill, and Callum Galbraith. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer-Verlag London, 1 edition, 2009. Available from: http://link.springer.com/10.1007/978-1-84882-406-5. xiii, 1, 6, 7, 10, 14, 15, 16, 19, 21, 22, 23, 24, 25, 26

[HF08] Annie Hui and Leila De Floriani. *Representing And Understanding Non-Manifold Objects*. PhD thesis, Maryland, 2008. 1

[Hu98] Tang Ming-Jye Hu. Dividing Cubes Algorithm, 1998. Available from: http://www.csee.umbc.edu/{~}ebert/693/THu/dividingcubes.html. 19

[HVFF13] John F Hughes, Andries Van Dam, James D Foley, and Steven K Feiner. *Computer graphics: principles and practice*. Pearson Education, 2013. 8

[HW90] Mark Hall and Joe Warren. Adaptive polygonalization of implicitly defined surfaces. *Computer Graphics and Applications, IEEE*, 10(6):33–42, 1990. 24, 25

[Ju09] Tao Ju. Fixing geometric errors on polygonal models: a survey. *Journal of Computer Science and Technology*, 24(1):19–29, 2009. 8

[KB04] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004. 8

[KS01] Tasso Karkanis and A James Stewart. Curvature-dependent triangulation of implicit surfaces. *Computer Graphics and Applications, IEEE*, 21(2):60–69, 2001. 9

[KTU+05] Kiwamu Kase, Yoshinori Teshima, Shugo Usami, Masaya Kato, Shuntaro Yamazaki, Masao Ito, and Akitake Makinouchi. CAD-CW-complexes based approach. *Computer-Aided Design*, 37(14):1509–1520, 2005. 26, 27

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987. vii, ix, 1, 15, 16

[Mea80] Donald J R Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980. 24

[Mea82] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982. 24

[Mor03] Richard Morris. A Client-Server System for the Visualisation of Algebraic Surfaces on the Web. In *Algebra, Geometry and Software Systems*, pages 239–253. Springer, 2003. 7, 54

[MSS94] C. Montani, R. Scateni, and R. Scopigno. Discretized Marching Cubes. *Proceedings Visualization '94*, 1994. 16

[Nat94] Balas K Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11(1):52–62, 1994. 18

[NY06] Timothy S. Newman and Hong Yi. A survey of the marching cubes algorithm. *Computers and Graphics (Pergamon)*, 30(5):854–879, 2006. 16, 18, 24

[RG06] Adriano N. Raposo and Abel J. P. Gomes. Polygonization of multi-component non-manifold implicit surfaces through a symbolic-numerical continuation

algorithm. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, pages 399–406. ACM Press, 2006. 1, 2, 10, 14

[RRS97] Angela Rösch, Matthias Ruhl, and Dietmar Saupe. Interactive visualization of implicit surfaces with singularities. In *Computer Graphics Forum*, volume 16, pages 295–306. Wiley Online Library, 1997. 8

[Shi97] M. Shiota. *Geometry of Subanalytic and Semialgebraic Sets*. Progress in Mathematics. Birkhauser, Boston, 1997. 45

[ST90] Peter Shirley and Allan Tuchman. *A polygonal approximation to direct scalar volume rendering*, volume 24. ACM, 1990. 20

[TALSZ14] Jean-Luc Toutant, Eric Andres, Gaelle Largeteau-Skapin, and Rita Zrour. Implicit digital surfaces in arbitrary dimensions. In *Discrete Geometry for Computer Imagery*, pages 332–343. Springer, 2014. 1

[VGdF07] Luiz Velho, Jonas Gomes, and Luiz H de Figueiredo. *Implicit objects in computer graphics*. Springer Science & Business Media, 2007. 1

[VW94] Allen Van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics (TOG)*, 13(4):337–375, 1994. 18

[WF07] Oliver Williams and Andrew Fitzgibbon. Gaussian process implicit surfaces. *Gaussian Proc. in Practice*, 2007. 1

[Wik15] Wikipedia. Octree — Wikipedia{,} The Free Encyclopedia, 2015. Available from: https://en.wikipedia.org/w/index.php?title=Octree{&}oldid=697546517. 24

[Wik16] Wikipedia. Marching tetrahedra — Wikipedia{,} The Free Encyclopedia, 2016. Available from: https://en.wikipedia.org/w/index.php?title=Marching{_}tetrahedra{&}oldid=714198492. 21, 23

[WKS86] Geoff Wyvill, Tosiyasu L Kunii, and Yasuto Shirai. Space division for ray tracing in CSG. *Computer Graphics and Applications, IEEE*, 6(4):28–34, 1986. 15

[WMW86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986. 16, 23

[WOK05] Jianning Wang, Manuel M Oliveira, and Arie E Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. In *Visualization, 2005. VIS 05. IEEE*, pages 415–422. IEEE, 2005. 26

[YKI02] Shuntaro Yamazaki, Kiwamu Kase, and Katsushi Ikeuchi. Non-manifold implicit surfaces based on discontinuous implicitization and polygonization.

In *Geometric Modeling and Processing, 2002. Proceedings*, pages 138–146. IEEE, 2002. xiii, 26, 27

[ZYP09] Wen Zheng, Jun-Hai Yong, and Jean-Claude Paul. Simulation of bubbles. *Graphical Models*, 71(6):229–239, 2009. 28

Glossary

Triangulation	Triangulation is a process of connecting surface points after sampling stage to obtain a triangular mesh representation of the surface.
Rendering	Rendering is a process of calculating the screen in a meaning that syntheses all desired components together and results in a final picture.
Sampling	The process of identify the location of an object in the space.
Polygonization	A generalized term of triangulation in which the generated mesh representing the surface can be composed of any type of polygons.
Geometry	Real shape of the objects representated by a mathematical functions.
Topology	A space that contents information about the components and their relationship of an object expressed by a mathematical function.
Interpolation	The process of estimating the value of a function at a point from its values at nearby points.
Polygonizer	A program that takes the defining equation, f(x,y,z)=0, of an algebraic surface and produces a polygonization of the surface [Mor03].
Mesh	The (mathematical) construction and computer implementation of an object, by defining points in a 3 dimensional array. This array is based on the X, Y and Z axis of geometric space. Then, different sets of these points are mathematically ”joined” by e.g. lines, to create polygons and the polygons joined to create objects. The simplest result is usually displayed a wireframe model. [FO03].
OpenGL	A 3D graphics programmer interface, initially design by SGI and now developed by several companies, to improve the performances of graphical hardward supporting the OpenGL standard. Direct3D (by Microsoft) is another standard implementation [FO03].
Shading	It is the assignment of surface properties to an object. They are colour, normal information, reflectance, transparency