Cubical Marching Squares Implementation

George Rassovsky

Master of Science
Computer Animation and Visual Effects
Thesis

September, 2014

Contents

List of Figures

List of Tables

List of Acronyms

ADF Adaptively-sampled Distance Field

BRep Boundary representation

BReps Boundary representations

BONO Branch-on-need Octree

BONOs Branch-on-need Octrees

FReps Functional representations

CSG Constructive Solid Geometry

CMS Cubical Marching Squares

MC Marching Cubes

DC Dual Contouring

MS Marching Squares

EMC Extended Marching Cubes

TMC Topology-Consistent Marching Cubes

MDC Manifold Dual Contouring

DMC Dual Marching Cubes

MRI Magnetic Resonance Imaging

MT Marching Triangles

MC33 Marching Cubes 33

SN Surface Nets

Abstract

Extracting a polygonal mesh from an existing scalar field has been a discussed topic over many years in the Computer Graphics field. Many algorithms have been developed which perform the task to a satisfactory level. But only a few claim to solve all known existing problems. However, due to greater complexitytheir usage never become widespread. One such algorithm is the Cubical Marching Squares (CMS) Ho *et al.* (2005), which claims to solve nearly all issues which other algorithms face. Now, a decade after its conception, any further research on it is scarce, and there are hardly any implementations. This thesis verifies its plausibility by developing an isosurface extraction library loosely based on a partial implementation of the CMS algorithm.

Keywords: Isosurface Extraction, Implicit Surfaces, Scalar Field Polygonisation, Meshing

Acknowledgements

I would like to especially thank Mathieu Sanchez for his help and supervision throughout the whole project. I want to also thank Dr. Oleg Fryazinov for his support, and Prof. Alexader Pasko for introducing me to the problem at hand and pointing me in the direction of this algorithm.

Chapter 1

Introduction

1.1 Problem statement

Visualisation of Volumetric Data

Functions of space are being generated by numerous sources, around us every day. From algebraic functions to medical apparatus, a myriad of areas could produce such functions or data sets of such functions. This data, more often than not, serves a purpose when visualised, thus the need for visualisation of volumetric data occurs. Two of the most prominent ways of representing such data are volumetric representations and Boundary representations (BReps).

The Computer Graphics problem which arises, related to this issue is the visualisation of such volumetric data. This document focuses on the extraction of an isosurface from a volumetric data set, into a polygonal mesh, which is a subset of Boundary representation (BRep). A polygonal mesh is a standard way of representing a geometric object by describing it's surface through a web of interconnected, approximating polygons. Thus it provides a robust way of visualising a volumetric data set by describing a gradient at a specific isovalue. Representing the volumetric data in that manner allows for efficient rendering, as modern day graphics hardware is highly optimised for working with such representations.

Applications

Medical Visualisation

One of the primary usages of an isosurface extractor is extracting a boundary representation from medical volumetric datasets. Medical visualisation has been used since the dawn of Computer Graphics. One can argue that it has pushed the field forward in many aspects, and has played a crucial role in the development of many of the algorithms known to us today. Medical visualisation refers to the visual representation of medical data. This data is in most cases a set of discrete samples in three-dimensional space, which produce a volumetric dataset, as in the case of Magnetic Resonance Imaging (MRI) scans. In most cases this data is rendered straight away with the help of a volumetric renderer, which usually produces a gray-scale image of the rendered region, Figure 1.1. Need exists that sometimes a boundary representation of a layer should be constructed. This is where an isosurface extraction algorithm will come into play and create a polygonal representation of a certain isolevel of the provided discrete scalar field. This model could be used for a detailed 3D view of individual layers or for a better polygonal rendering of certain organs, tissues, or any object of interest, which could be extracted from within the data set.

Contouring an algebraic scalar field

Sometimes, functional representations Functional representations (FReps) are preferred over BReps, due to the ease of performing geometric operations. Such an approach provides a continuous function of space, which could be used as a scalar field and a polygonal mesh could be extracted at any isovalue. Equation 1.1, is an implicit equation of a sphere of radius R, which is defined at the isovalue of 0.

$$x^2 + y^2 + z^2 - R^2 = 0 \tag{1.1}$$

Point cloud reconstruction

Reconstruction of a mesh from an existing point cloud is a field in its

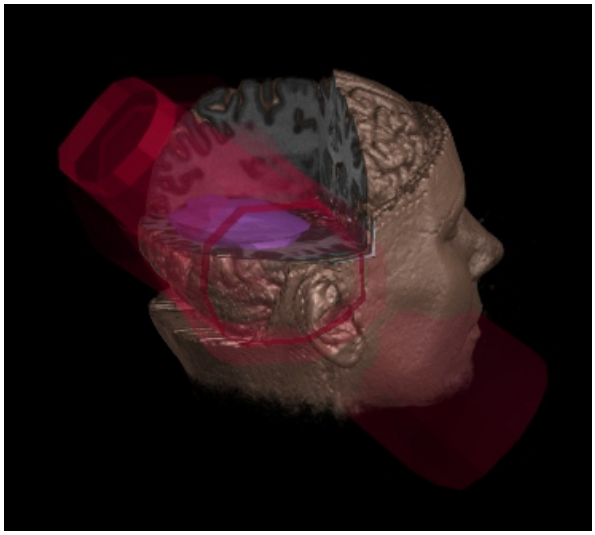

Figure 1.1: *Volumetric render of medical data. Image source: Levoy (2006)*

own right. There are many techniques that deal with the topic. A good and robust isosurface extractor would be able to tackle this problem too. It is possible to do so by a conversion of the point cloud to a scalar field. From there on the algorithm would run as usual to produce a polygonal mesh from the point cloud scalar field, Figure 1.2.

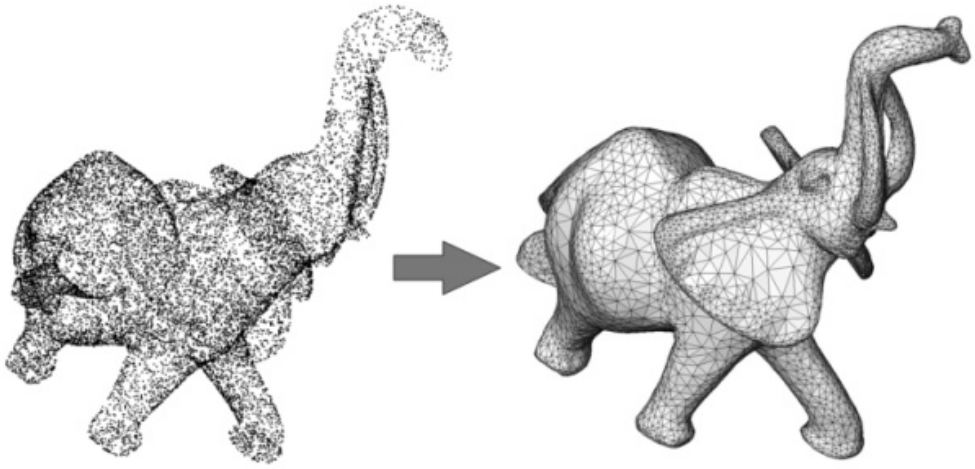

Figure 1.2: *An elephant point cloud reconstructed CGAL's point cloud reconstruction algorithm. Image sourced: CGAL (2014)*

Mesh retopologisation

Some procedurally derived meshes come with redundant geometry or bad topology. In the first case, there might be too much polygons, where the

detail of the model could be described with fewer. Figure 1.3, shows the simplification of a 'Stanford Bunny' model into half of it's original poly-count, whilst preserving most of the detail. In the case of bad topology, which can be caused by a number of reasons, as for example, very disproportional triangles, which result in heavier computation and greter error when rendered. An example of such a mesh, which is often obtained from the polygonisation of letters in 3D Software packages can be seen in Figure 1.4. Both of the cases described above could be handled with an adaptive isosurface extraction technique, which would produce a mesh with consistent topology and reduce the amount of polygons to the bare minimum, based on its reduction condition, whilst preserving the detail of the original mesh.

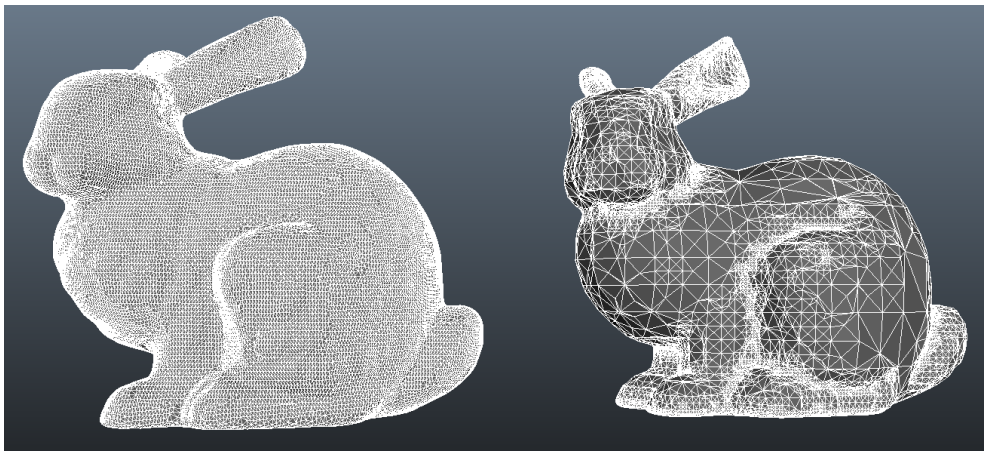

Figure 1.3: *A stanford bunny model, consisting of 69,666 polygons (left), and a simplified mesh with 34,350 polygons (right). 3D Model Courtesy of: Stanford (2013).*

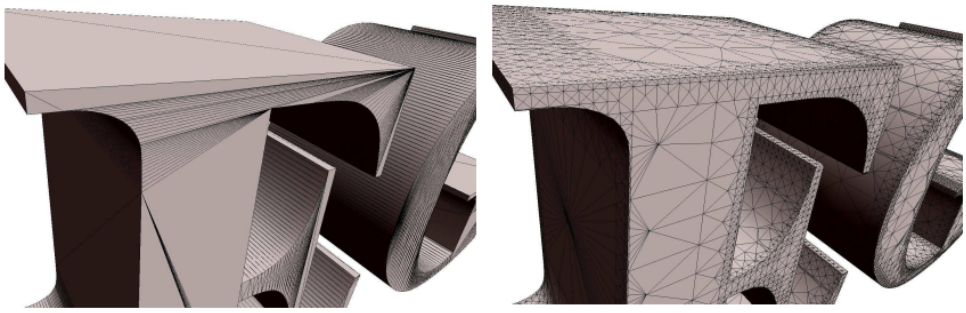

Figure 1.4: *Retopologisation of meshes of letters, with bad topology. Image sourced: Ho* et al. *(2005)*

Problem

There are vast amounts of methods which aim to tackle the problem of isosurface extraction. Each of these methods has its pros and cons, and they are often used as is appropriate for the task at hand. Some extract meshes with too much polygons, even where they are not necessary, some lose any sharp features or small detail in the underlying surface, others suffer from topological inconsistencies, self-intersections, inter-cell dependencies, etc.

A robust and standard way, which handles all of those problems is still not truly developed and agreed upon. There are however a few candidates, which have high claims, one of them is the CMS algorithm Ho *et al.* (2005). Now, nearly ten years since its official debút, this algorithm's plausibility, still remains a mystery to a vast majority of the Computer Graphics world. The fact that, as of this date, there are very few actual implementations or even descriptions of the algorithm, to be found in the public domain, despite its problem solving claims, speaks for itself. Apart from the authors' original source code, during this research, only one partial implementation of the algorithm was found. It is located in the libfab C library of the 'Kokopelli project' Keeter (2013).

1.2 Objectives

The goal of this project is the creation of a robust stand-alone C++ library, which could be extended and improved in the future. The library would contain the core of the CMS algorithm, so that it can put its claims to the test. This project is limited to implementing adaptive and manifold meshing with the CMS algorithm, without any additional post-processes, such as crack-patching. Based on such an implementation all other features considered by the original algorithm, would eventually be possible to accomplish without the need for cumbersome combination with other algorithms or techniques. The core algorithm, should produce meshes topologically equivalent to Marching Cubes, however a successful

implementation of the CMS idea, would allow for adaptive resolution based on a user controlled heuristic.

The application created should take in a discrete scalar field of some type, and then extract a polygonal mesh representing the input field at a specified isolevel. The generation of the function of space, is beyond the scope of this project. It is rather provided by the user, as an actual algebraic function or some type of scalar or Adaptively-sampled Distance Field (ADF), which allows it to be sampled or interpolated at any point in space.

1.3 Structure

Chapter 2: Literature Review provides a overview of the previous work in the field of isosurface extraction, beginning at the dawn of Computer Graphics and leading up to nowadays. It portrays the cause-effect relationship between the different techniques, highliting their downsides and respectively their solutions.

Chapter 3: Technical Background describes, in brief, the way isosurface extraction algorithms, of the spatial partitioning type, work, starting with the Marching Squares algorithm and then expands the idea into 3D as in the original Marching Cubes. Then builds on that foundation with the Cubical Marching Squares algorithm in greater detail.

Chapter 4: Implementation takes a detailed approach to describing the main steps which were taken during the loose implementation of the Cubical Marching Squares algorithm in this project.

Chapter 5: Results and Discussion displays some of the acquired results and comparisons. Then forms a discussion based upon the results and comparisons obtained from the program created in this project.

Chapter 6: Conclusion continues with final remarks on the discussion started in the previous chapter, drawing conclusions from the

outcome of the project. Then describes possible future work, in terms of additional features, optimisation and algorithm expansion.

Chapter 2

Literature Review

2.1 Visualisation of Implicit Surfaces

> "*An implicit surface* $\{\boldsymbol{p} \in \mathbb{R}^3 : f(\boldsymbol{p}) = 0\}$ *is a two-dimensional manifold, provided that* f *is continuous and 0 is a regular value of* f *(that is, the gradient is defined for all points* $\boldsymbol{p}$ *on the surface). This means that the surface may be triangulated.*"
> (Bloomenthal and Wyvill (1997), p.128)

Implicit surfaces are surfaces defined by a specific value of a scalar functions in 3D space. They, therefore, could be expressed as single equations with three variables. The visualisation of such surfaces has been a matter of discussion over the years, ever since the dawn of Computer Graphics.

There are two main ways to visualising Implicit Surfaces. The first is the incorporation of ray tracing techniques for the direct rendering of implicit surfaces Hanrahan (1983) or even directly of Constructive Solid Geometry (CSG) trees Wyvill *et al.* (1986). The second method is the polygonisation of an implicit surface (isosurface), which then allows for rendering of a polygonal mesh. Both schemes have been extensively researched over the years and have been immensely used, incorporated in a large amount of software products. They both have their pros and

cons. In simple terms, if precision and minimal visual error are of higher priority, the ray tracing technique would be preferable, however due to it's ray-tracing nature, it would come at a higher computational cost. Furthermore rendering in real-time, depends greatly on the isosurface at hand. Thus it cannot be guaranteed even with the hardware solutions of this day. On the other hand, if robustness, and real-time visualisation are key, a technique revolving around the extraction of the implicit surface as a polygonal mesh, could be relied upon. This extracted surface can then be visualised using standard online or offline rendering techniques and could be manipulated using polygonal editing techniques.

2.2 Isosurface Extraction

Isosurface extraction is not a new problems in the field of Computer Graphics. The first attempts to tackle it date back to the early 70s of the *XX-th* century, with researches such as Keppel (1975), on IBM's side. However it is in the last two decades of the same century that work with it really begins. MC Lorensen and Cline (1987) is probably, one of the most well-known algorithms in Computer Graphics and by far the most cited resource in the field, according to the ACM Digital Library ACM (2014). Lorensen and Cline lay the foundation for what would become, one of the most prominent technique for extracting a polygonal mesh from function of space, widely used in it's 'original' form even to this day. This led to a great number of techniques virtually based on MC's principles of spatial partitioning.

When it comes to isosurface extraction there are a few distinct methodologies, which take a very different approach to the same problem. They can be classified into three main groups:

Spatial Partitioning

(or Spatial Sampling) (e.g. MC) All such algorithms, have in common, that given a constrained domain in 3D space, they begin by applying spatial decomposition on that domain into sub-domains (also referred to as: cells, cubes, voxels, etc.), discarding sub-domains which do not contain (intersect, straddle) the isosurface. These algorithms then proceed, to reproduce the original isosurface by approximating it with polygons in every surface-containing sub-domain. Figure 2.1, demonstrates the general concept with a 2D example.

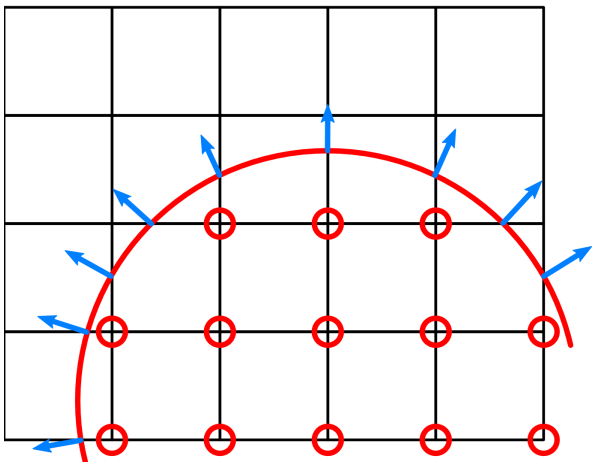

Figure 2.1: *A 2D variant of a 3D spatial partitioning isosurface extraction algorithm on a regular grid. The red curve being the isosurface, the latice represents the regular grid, and the red circles denote the corners of the 'voxels' which are considered inside the surface. Image source: Upvoid (2014)*

Surface Tracking

(or Continuation methods) Surface Tracking methods could be grouped into two classes based on the approach they take. *Cellular approaches* Allgower and Gnutzmann (1991), given an initial surface straddling cell as input procedurally find other such neighbouring cells, thus tracking down the surface. They share all the problems of classical Spatial Sampling algorithms. *Delaunay-based* or *Particle* approaches, Szeliski and

Tonnesen (1992), Witkin and Heckbert (1994), are certainly more famous of the two. They generate particles on the boundary of the implicit surface and then polygonise the particles to create the approximating mesh. In most cases, Delauney triangulation is employed, for that second part of the algorithms, hence the name. One such algorithm is the original Marching Triangles (MT) technique Hilton *et al.* (1996), along with its more recent improvements such as adaptive meshing Akkouche *et al.* (2001). Figure 2.2, shows two of the stages of the MT algorithm and how the particles on the surface are polygonised, while Figure 2.3 demonstrates the algorithm in action, progressively meshing a spherical object.

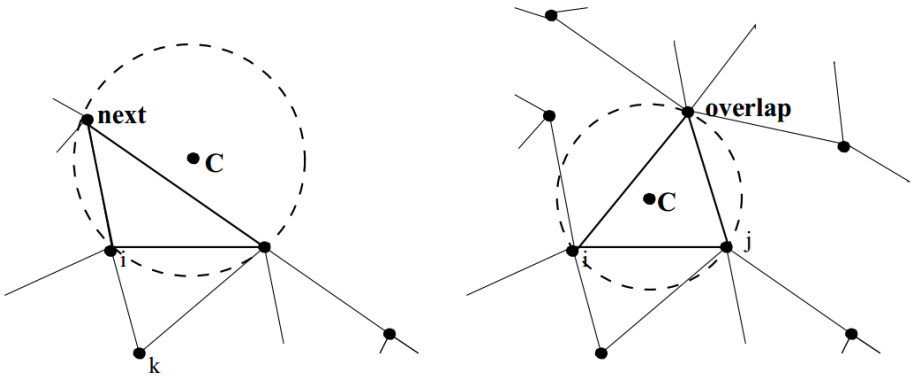

Figure 2.2: *Stages of the Marching Triangles algorithm, where the surface is tracked down using Delaunay constraint, and searching for an overlapping particle. Image source: Hilton* et al. *(1996)*

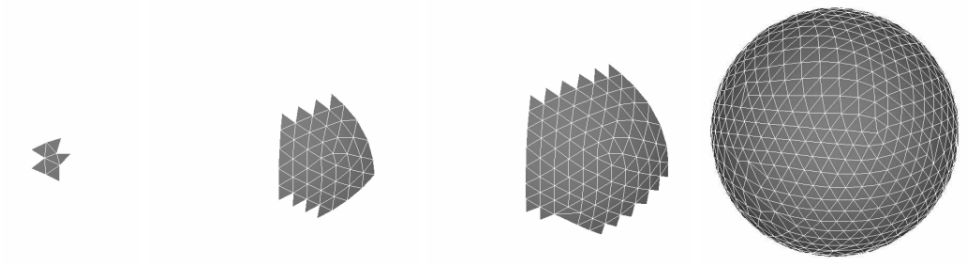

Figure 2.3: *The Marching Triangles algorithm sequentially generating the mesh, approximating the surface. Image source: Hilton* et al. *(1996)*

Surface Fitting

Surface Fitting techniques sequentially approximate an initial 'seed mesh' to the implicit surface. Such algorithms could be further classified under two main types. The first being the *Element Driven* approaches Desbrun *et al.* (1995) and Crespin *et al.* (1998), which provide a base mesh enclosing for each primitive, then approximate the surface and combine all resultant meshes into one global one. This technique offers efficiency and robustness due to it's hierarchical approach, however, it also suffers from many drawbacks, of which, possible inconsistent topology and wrong tessellation, are only some. Figure 2.4 demonstrates the main stages in the polygonisation of implicit sweep surfaces algorithm.
Shrink Wrap approaches, as the original technique Overveld and Wyvill (1993) and an improved version which handles arbitrary geometry Bottino *et al.* (1996), on the other hand, are global, unlike the element driven ones. A base mesh is supplied surrounding the implicit surface, this mesh then systematically converges towards the implicit surface. This procedure results in loss of any concavity details. Further improvements on the technique add so called, critical points which are established on the boundary of the implicit surface, so that such details are detected and preserved.

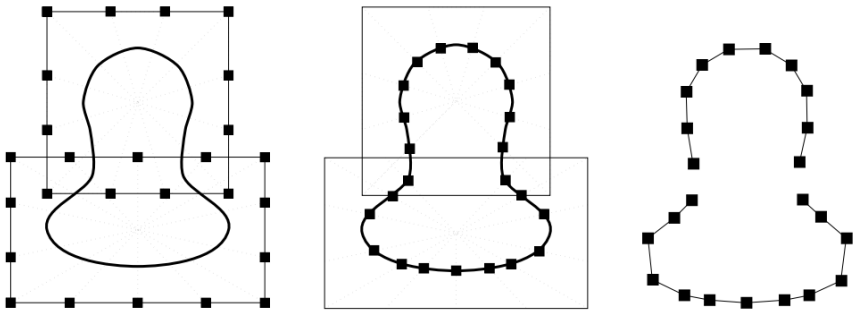

Figure 2.4: *A surface fitting technique, demonstrating the three stages of production: (from left to right) (a.) Initialization, (b.) migration and validation and (c.) the output of the final mesh. Image source: Crespin* et al. *(1998)*

Comparisons

There is a tendency for comparisons between techniques belonging to differing categories of the ones specified earlier. In many cases such comparisons could be ambiguous and the conclusions drawn could be deceiving, as techniques of different categories do not pertain to one another. This comment is applicable in most cases, unless the goal is a general observation on the topic of isosurface extraction. This project tackles the problem of isosurface extraction based on a spatial sampling algorithm, therefore it will focus on comparisons and a deeper investigation into precisely such methods. Further in-depth analysis and comparisons with Surface Fitting or Surface Tracking techniques fall beyond the scope of this project. Figure 2.5 depicts the hierarchical structure of the field, highlighting the branch which this project occupies, for maintaining clarity.

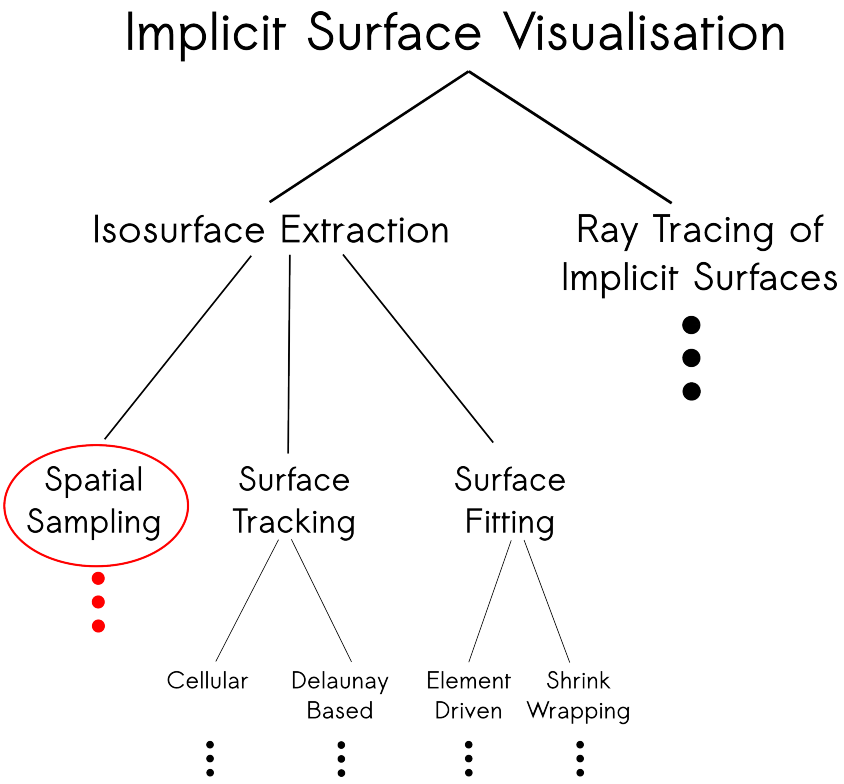

Figure 2.5: *A hierarchical graph visualising the branch of isosurface extraction that this project focuses on.*

2.3 Spatial Sampling Algorithms

Many derivatives of the original MC algorithm, will preserve the voxel-based approach pioneered by Lorensen and Cline. The algorithm is intuitive and relatively easy to comprehend, degenerating an infinite amount of surface variations into 256 possible test cases which, using symmetry, could be even further simplified into 15 general patterns, as seen in Figure 2.6. All this comes at a relatively low-cost compared to previous techniques and even more so, a much lower-complexity. According to later evaluation of the original algorithms, the original MC, suffered from three main issues, *topological inaccuracies*, *cracks at adaptive resolution* and *loss of sharp features*, Ho *et al.* (2005). In the next sub-sections those problems would be addressed as well as the work done to address and solve them.

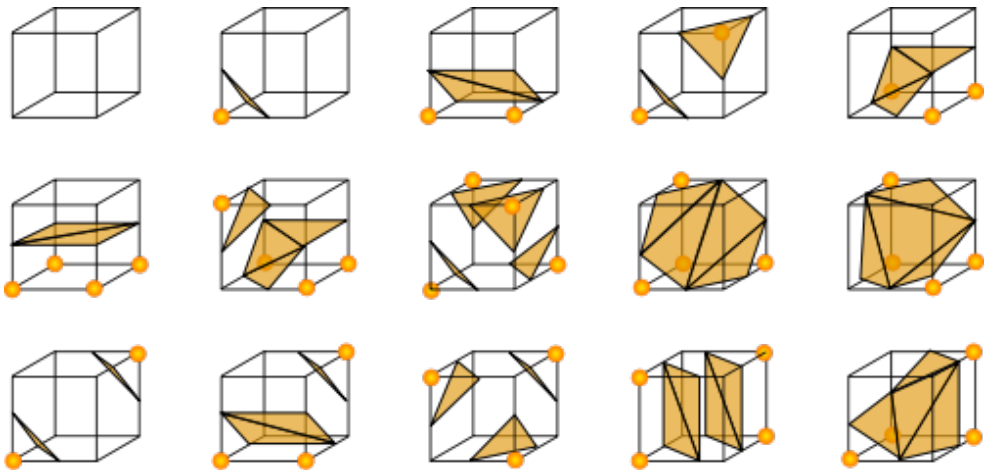

Figure 2.6: *The 15 configurations into which the 256 variations of straddling cubes, decompose, in the MC algorithm. Image Source: FountainComputer (2014)*

Holes - Inaccurate Topology

Inaccurate topology in MC is caused due to *face ambiguities*, see Figrure 2.8 and *internal ambiguities*, see Figure 2.10. They can cause the mesh to be non-homeomorphic to the surface, however handling these ambiguities in a wrong manner could result in the mesh being non-manifold, due to

holes. Holes in a mesh polygonised with the original MC, were a direct result of such face ambiguity.
When considering the problem in 2D Marching Squares (MS), there are two configurations, which could result in ambiguous topology. That is to say, that the decision of which case should be used is undefined by the algorithm. Those cases, for the MS algorithm are visible in Figure 2.7.

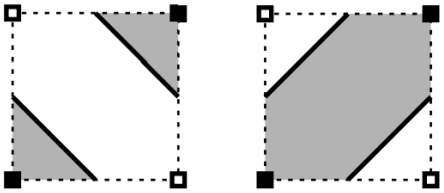

Figure 2.7: *The black corners are inside the surface. For this specific case in 2D there are two possible configurations. This is then considered a face ambiguity. Image Source: Ho* et al. *(2005)*

Face Ambiguity

In 3D the problem remains, and even gets worse, as 6 out of the 15 topologies could potentially end up creating holes, Figure 2.8 visualises the problem.

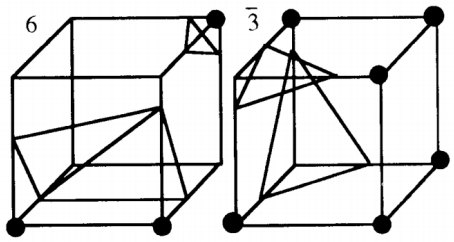

Figure 2.8: *A hole that will occur due to face ambiguity, on the face between those two cubes, formed with inconsistent usage of the original MC algorithm. Image Source: Nielson and Hamann (1991)*

The easy workaround for this problem, which does not guarantee topological correctness, with regards to the isosurface, however guaranteeing that the surface would be without holes, is being consistent throughout the meshing. Being consistent involves tessellating always in the same manner for each specific case. This would avoid holes generated by face ambiguities.

More complex solutions were later developed which overcame this problem altogether. They would guarantee tessellation accurate to the geometry with each choice of configuration, at the same time preserving consistent topology. The Asymptotic Decider Nielson and Hamann (1991), is one of them. It decides by making a comparison between the surface value and the value acquired from the bilinear interpolation of the intersection point at the asymptotes, across a face.

Other schemes dealing with disambiguation of the original MC, involve a modified lookup table, Montani *et al.* (1994), *gradient-based methods* Wilhelms and Van Gelder (1990), *feature-based methods*, a technique referred to as *prefered polarity* Bloomenthal (1988) and *cell decomposition*, refered to as, *domain tetrahedralization* Bloomenthal (1988), Payne and Toga (1990), where tetrahedra are used, which is covered in greater depth below. Some publications which offer a more detailed account on topological disambiguation in MC-based isosurface extraction are Ning and Bloomenthal (1993) and Newman and Yi (2006).

Face ambiguities could be avoided altogether by adopting different *cell decomposition* or simplical decomposition strategies. In many cases the base shape of decomposition is a tetrahedron, an idea borrowed from molecular modeling publications such as Koide *et al.* (1986), which was later incorporated in isosurface polygonisation techniques, as described in Bloomenthal (1988) and Payne and Toga (1990), see Figure 2.9. Others decompose into sub-cubes, Wilhelms and Van Gelder (1990). CMS, Ho *et al.* (2005) also decomposes a cell into six 2D quads, however it is not solely for disambiguation purposes and a quad is technically not considered a simplex, therefore CMS does not fall into this category of algorithms. Simplical decomposition methods often result in meshes with more vertices than a standard MC-based cellular algorithm. The increase in vertices is on average by a factor of two Ning and Bloomenthal (1993).

Internal Ambiguity

Internal ambiguities, are an extension of the face ambiguities, into three-dimensional space. However, solving face ambiguities does not automat-

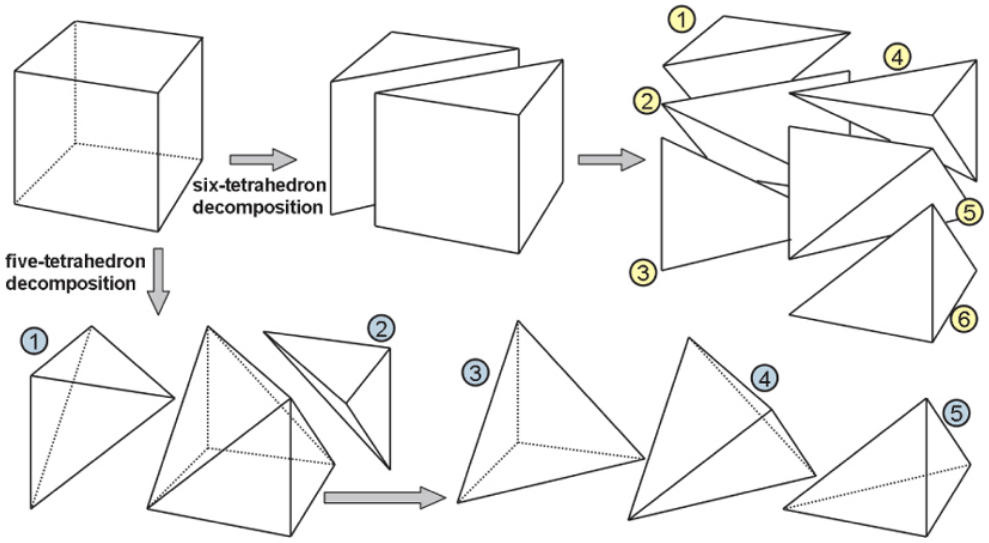

Figure 2.9: *The two most common ways of decomposing a cube into tetrahedra. Image Source: Chu* et al. *(2009)*

ically solve internal ambiguity. This is the case in Figure 2.10, where the surfaces formes a cylindrical structure within the cell. This issue has to be dealt with separately as even a face disambiguator would fail to locate it.

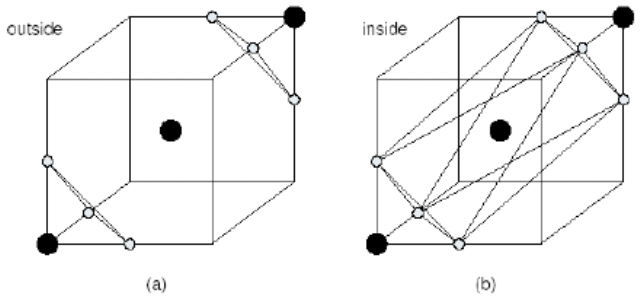

Figure 2.10: *A case of internal ambiguity and a solution to it with an extra sample point in the centre of the cell. The sample result is outside the surface in (a) and inside the surface in (b). Image Source: Tsuzuki* et al. *(2007)*

Internal ambiguities were dealt with by two independent researches producing similar results Nat (1994) and Marching Cubes 33 (MC33) Chernyaev (1995), who proposed a new, thirty-three-case table to replace the fifteen-case table of the original MC, and solve the internal ambiguity issue in three basic steps.

In 2003, a full implementation of Chernyaev's method, under the title "Efficient implementation of Marching Cubes' cases with topological guarantees" Lewiner *et al.* (2003) provided a topologically consistent implementation of MC, dealing with all topological issues, Figure 2.11.

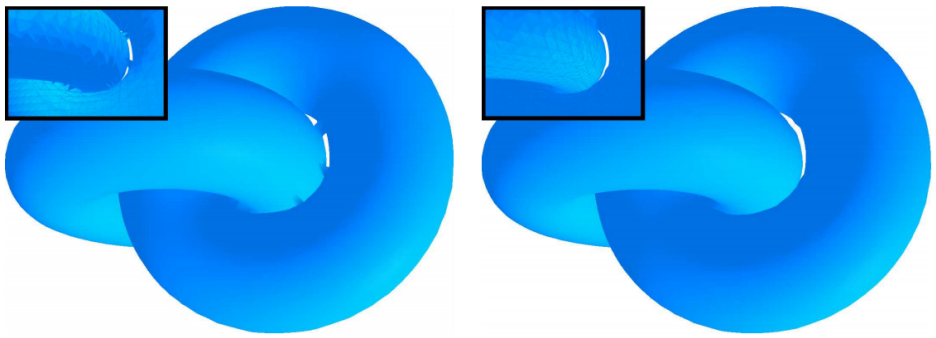

Figure 2.11: *The image depicts two linked torii, original MC on the left and the topologically consistent implementation of the right. The left part of the figure showcases the artifacts in the mesh caused by the ambiguities, resulting in topological error, to the underlying isosurface. The right side is the disambiguated solution. Image Source: Lewiner* et al. *(2003).*

All those techniques are based on the original MC and deal with topological inaccuracy in the original MC. For clarity, those methods which deal with this issue in the original MC, would be referred to as Topology-Consistent Marching Cubes (TMC), through the rest of this document.

One thing that must be considered is that **topological correctness** does not necessarily mean **topological equivalence** with the surface. Topological equivalence, guaranteeing that the mesh would be homeomorphic to the surface requires additional information about the underlying scalar field, as stated by Etiene *et al.* (2012).

Cracks in Adaptive Resolution

> *"When the cells are fixed in size, a* ***fixed resolution*** *(or uniform space) partitioning occurs. When the cell size is locally proportional to the size of surface detail, an* ***adaptive resolution*** *partitioning occurs."* (Bloomenthal and Wyvill (1997), p.129)

Due to the nature of the original MC algorithm, and its sampling on a regular grid, it has a Nyquist-like limit and suffers from classical signal processing problems such as *under-sampling*, in which case detail is lost, and *over-sampling*, which results in redundant sampling and unnecessary data storage, as the portion of the surface could be represented correctly within less samples. Under-sampling the underlying field function results in three-dimensional aliasing.

Types of adaptive datastructures

Shortly after the initial MC algorithm, sampling on adaptive structures was introduced, resulting in a more optimised sampling process and as a positive consequence - simplified meshes Bloomenthal (1988), such algorithms rely on hierarchical data-structures to expand and explore only regions of interest to the algorithm. This interest could often be classified simply as surface detail. Different algorithms take different approaches to the conditions of subdivision of space. Some of the most popular adaptive data-structures used in the field include *Adaptive Octrees* Bloomenthal (1988), Shu *et al.* (1995), Ju *et al.* (2002), Ho *et al.* (2005), *Restricted Octrees* Westermann *et al.* (1999), Kazhdan *et al.* (2007), Branch-on-need Octrees (BONOs) Wilhelms and Van Gelder (1992), and others.

Generation of the Adaptive Structure In most cases the structure is constructed using cubic cells, resulting in cuboidal structures (octrees). Other non-cellular methods for subdivision exist. The next most wide-spread type is tetrahedral subdivision Hall and Warren (1990). Note that not all tetrahedral adaptive algorithms, subdivide tetrahedraly, but rather cubically, and then decompose the cube into tetrahedra

Bloomenthal (1988).

Different techniques, subdivide based on different criteria when creating the adaptive structure. Some check for intermidiate surface intersections, or high divergence of the surface normals at the corners of the checked cell. Whilst others rely on the anticipated polygonisation of the surface in that cell and subdivide if it results in too many polygons, if a polygon centre is at a low proximity to the surface, or based on the surface normals at the polygon vertices. Other more sophisticated analytical ways of checking for potential complex surface within the cell, exist which are based on interval arithmetics Duff (1992).

Adaptive resolution could be achieved with another approaches, one of which being *iterative surface refinement*. In that technique it is not the spatial data structure which gets subdivided, but rather, the actual polygonal representation gets iteratively refined at places which require more polygons in order to preserve surface detail, as such approaches normally commence with a course polygonal representation of the surface, Allgower and Gnutzmann (1991).

Cracks problem

The adaptive feature, leads to the second problem, namely cracks at adaptive resolution, as stated earlier. They occur when two adjacent cells have different resolution, Figure 2.12. Some techniques avoid them altogether by incorporating different methods. Bloomenthal (1988) and Ning and Hesselink (1991), make the larger cell use the edges of its smaller neighbours, whilst Mullner and Jablokow (1993) combines vertices based on their spatial proximity, instead of their edge membership. This however might not always produce correct results, and comes at the expense of accuracy. Hall and Warren (1990) propose the *honeycomb* method, which avoids cracks, using tetrahedral subdivision, in a manner which does allow for the propagation of the subdivision throughout the spatial partitioning.

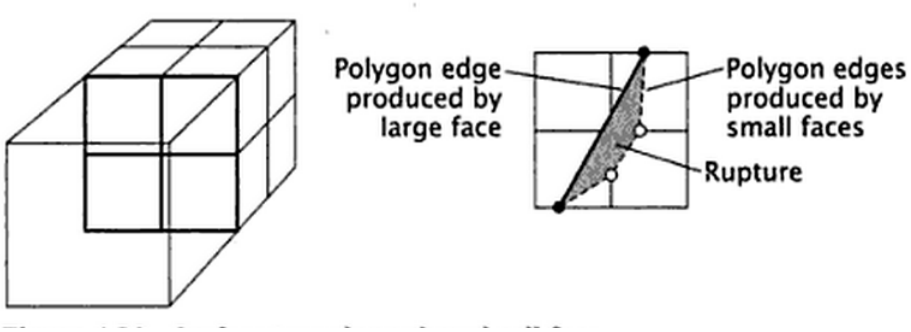

Figure 2.12: *A crack on a shared face, between cells of different subdivision, Bloomenthal and Wyvill (1997).*

One wide-spread variety of techniques for dealing with the cracks, involves a post-process known as crack-patching. It is done in different ways by different algorithms. Shu *et al.* (1995), locate and polygonise the cracks, which however, results in a G^1 discontinuity in the surface. Crack-patching is usually a heavy and undesired post-process therefore many algorithms, try and avoid it by dealing with surface ruptures during the main stage of the algorithm. CMS Ho *et al.* (2005) avoid cracks by using the data from sub-faces on shared faces between cells of different subdivision.

Updates and improvements to those methods have been proposed over the years. Some offer similar approaches, differing in the generation of the octree. As some would create the octree to its final level and then decimate the cells which are not surface straddling Shekhar *et al.* (1996). Others achieve adaptiveness by merging smaller cells, into bigger ones, if they lack sufficient detail Ning and Hesselink (1991).

Sharp Feature Preservation

The problem with loss of sharp features was not tackled until much later, compared to the other problems considered so far. Sharp features were preserved in the, so called Extended Marching Cubes (EMC), Kobbelt *et al.* (2001), which managed to do so, by storing additional information. Along with the plain scalar field data, it also stores normals to the surface. It then detects and samples sharp features in cells and

performs an edge flipping operation in order to preserve sharp features. An year later, Ju et al. develop the Dual Contouring (DC) algorithm Ju *et al.* (2002), inspired by the additional data which Kobbelt *et al.* (2001), were storing, which they name, *Hermite Data*. DC also borrows from Sarah Frisken's previous dual method called Surface Nets (SN), Gibson (1998), which uses numerous cells to create individual polygons. This feature, which is common for all 'dual' methods, however, gives rise to another problem, namely *inter-cell dependency*, which does not allow for the algorithm to be truly 'parallel', Ho *et al.* (2005). Furthermore, the original DC, produces self-intersecting meshes, which are therefore non-manifold. This problem is dealt with in the publication Manifold Dual Contouring (MDC), Schaefer *et al.* (2007). On the other hand the MC approach has also been used along with a dual approach in order to preserve sharp features, Nielson (2004).

Those are some of the main publications and techniques derived as a 'direct consequence' of the original MC. Figure 2.13, provides a visual overview of the chapter, with the relationships between the techniques and their influence.

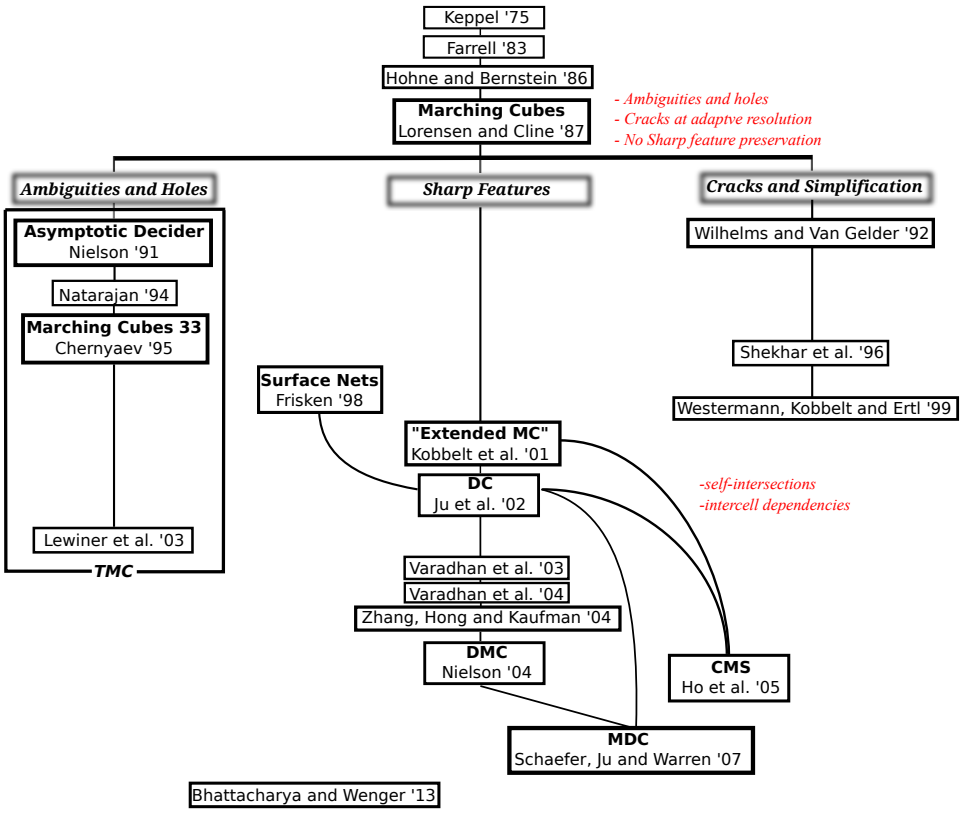

Figure 2.13: *A visual overview of the techniques discussed in this chapter, their relationships and influences are also depicted.*

Chapter 3

Technical Background

The goal of this project was, investigating into the field of robust polygonisation of a scalar field and creating a C++ library for isosurface extraction. Some of the proposed features, towards which this project was aimed are: producing manifold (watertight) meshes e.g. no holes, cracks or self-intersections, and meshing at adaptive resolution. From the spatial partitioning algorithms, there are few which claim those, and other, features. They usually consist of complex combinations of previously existing algorithms. Some algorithms that are worth mentioning are, Dual Marching Cubes (DMC) Nielson (2004), MDC Schaefer *et al.* (2007) and CMS Ho *et al.* (2005). There are some others, which claim more or less similar results to theirs, but those mentioned stand out from the rest. They all have their pros and cons, and non of them is perfect. This project is loosely based on the latter, CMS algorithm, as it takes an interesting approach, which will be covered in greater detail below. DMC and MDC are both combinations of variations of the MC Lorensen and Cline (1987) algorithm and one of the earliest 'dual' algorithms, SN Gibson (1998) as well as the later DC Ju *et al.* (2002).

On the other hand, CMS stays loyal to the original MC approach, however deals with some of its downsides, by simplifying the problem to squares instead of cubes. Therefore the original MC is decimated into six MS algorithms. This results in a much more complex algorithm, as the cells not only have to be decomposed but also have to be reconstructed

in order to produce the end result, however in the midst of this process many of the problems with the original MC can be addressed.

3.1 Marching Squares

Before proceeding it is important that some of the main concepts of the foundational techniques are briefly described. MS is a special case of the MC algorithm, restricted to two-dimensional space. Therefore it is used for the extraction of isocurves and isolines. $\mathbb{R}^2$ space is sampled on a regular grid $f(x, y)$, each square is defined by four of those sampling points, as denoted in Figure 3.1.

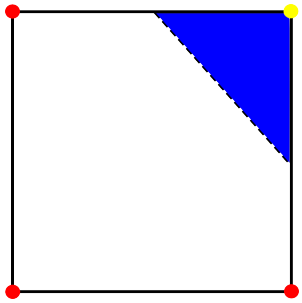

Figure 3.1: *A marching square. The points at the corners denote the sample points. In this example, red are outside the isoline (or isoband) and yellow inside the isoline. The dotted line, marks the isoline, and the blue colour defining the 'inside'.*

Each sample could be either inside or outside the surface, based on the value that the function has returned. In this thesis the signs are defined as negative inside and positive outside, however this could vary between different implementations. The four corners of each square provide $2^4 = 16$ possible configurations, Figure 3.2.

The surface then is approximated by creating lines between the edges corresponding to the correct configuration in each cell which crosses the isoline. There are two cases from those sixteen in Figure 3.2, which are 'ambiguous' as they could potentially produce different line results. This is known as a face ambiguity. There are techniques which deal with the disambiguation of face ambiguities, however, staying consistent

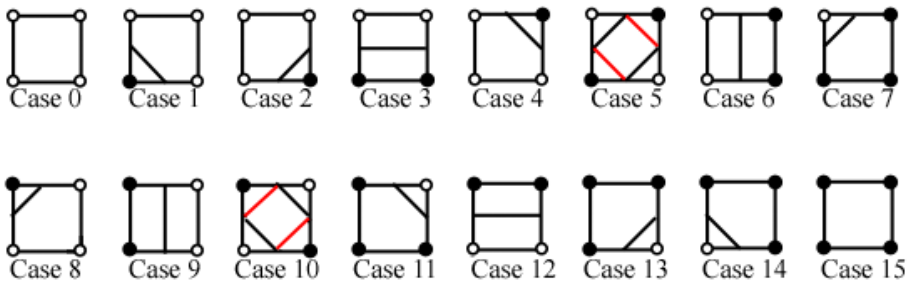

Figure 3.2: *The 16 configurations of the Marching Squares algorithm. Image Source: Gervaise and Richard (2002)*

in a concrete line configuration, guarantees a curve which would not be ruptured. These sixteen cases could further be simplified into only four unique cases, using symmetry and mirroring, one of which would have the mentioned face ambiguity, Figure 3.3.

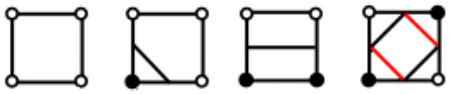

Figure 3.3: *The 4 unique configurations of the Marching Squares algorithm, which are necessary to reproduce all others. Image edited from: Gervaise and Richard (2002)*

When a configuration is chosen and the edge(s), between each two sampling points, which the surface is crossing, are discovered, it is possible to further enhance the approximation by finding the exact crossing point. This could be done by finding the isovalue-crossing point of the isocurve on the edge, which is the exact crossing point of the isocurve on that particular edge. This is often done by iterative linear interpolation along the edge, closing down on the surface, until a certain threshold is reached or a limit in the iterations is hit. This technique produces results as seen in Figure 4.16, which produce a better approximation of the isocurve. The 3D equivalent of using this approximation method can be seen in the comparison on Figure 4.17.

3.2 Marching Cubes

Marching Cubes and Marching Squares are essentially the same algorithm in different dimensions. Therefore the backbone of both algorithms is exactly the same. They only differ on implementational details, proceeding from the difference in dimensionality. MC, therefore, extracts an isosurface rather than an isocurve or isoline and works with a provided 3D scalar field. Hence, it uses cubes, rather than squares. Because each cube has eight corners the possible configurations become $2^8 = 256$, which can be further simplified to fifteen, as in the original MC Lorensen and Cline (1987), Figure 2.6.

3.3 Cubical Marching Squares

The CMS algorithm, bases its structure on the MC algorithm, but improves it in many ways. Some of the major downsides of MC were covered in the Literature Review section 2. Most of them are addressed by CMS as well as other issues, which have arised from other developments based on the original MC.

Input

The original CMS algorithm Ho *et al.* (2005) starts by taking input data and converting it into a unique standard format. The input could vary from polygonal meshes, point cloud sets, scalar fields, ADFs, or an algebraic functions. Any input is then converted into the standard type, which is Hermite Data, first employed for the purpose in Isosurface Extraction, by EMC Kobbelt *et al.* (2001) and DC Ju *et al.* (2002). Hermite data is gathered on a signed grid with exact intersection points (laying on the surface) and the normals at the points (sample points and sample normals).

Construction of the Signed Adaptive Octree

The adaptive sampling process in CMS starts with sampling a course uniform grid up to a certain level, which then continues to be subdivided and sampled until either, there is no need for subdivision, or the maximum level of the octree is reached.

The adaptive subdivision of cells is done based on two conditions, *edge ambiguity* and a potential *complex surface*.

- The Edge Ambiguity check makes sure there is a single sign change on any cell edge, expressing a sign change. If this check fails, there is an edge ambiguity and the cell to which this edge belongs, must be subdivided.
- The Complex Surface checks if there is a tendency that the cell might contain a 'complex' surface, which is detected by a heuristic based on the sample normals on that face. If the angle between the normals is above a certain, predefined threshold, the cell should be subdivided.

In CMS, faces are subdivided first, before the actual cell is subdivided and the relationships between sub-faces and sub-cells is stored for reference.

Generating Segments on Leaf Faces

After the signed adaptive octree is constructed, the algorithm proceeds with the generation of segments on every leaf (surface containing) face. This is done by running the MS algorithm for every such leaf face. Because of the fact that the simplified MS table also has an ambiguous case, the face disambiguation process is performed at that stage by checking for overlapping face sharp features based on the sample normals, as could be seen in Figure 3.4. The face sharp features detection is done by finding the intersection point of the tangent lines that the two sample points define with their sample normals. The exact intersection point is recorded

and a new segment is generated to reflect the face sharp feature. Figure 3.5, visualises the face segments and their combination in a single cube.

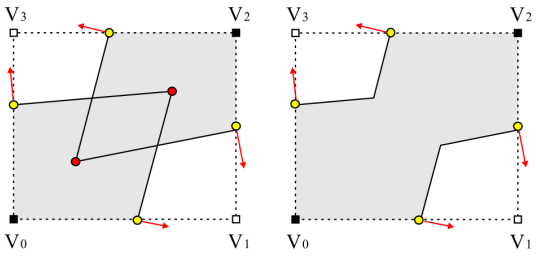

Figure 3.4: *A cell face which has intersecting sharp features. Self-intersection should not happen in a volume, therefore the algorithm chooses the correct disambiguated configuration, with preserved sharp features. Image source: Ho* et al. *(2005)*

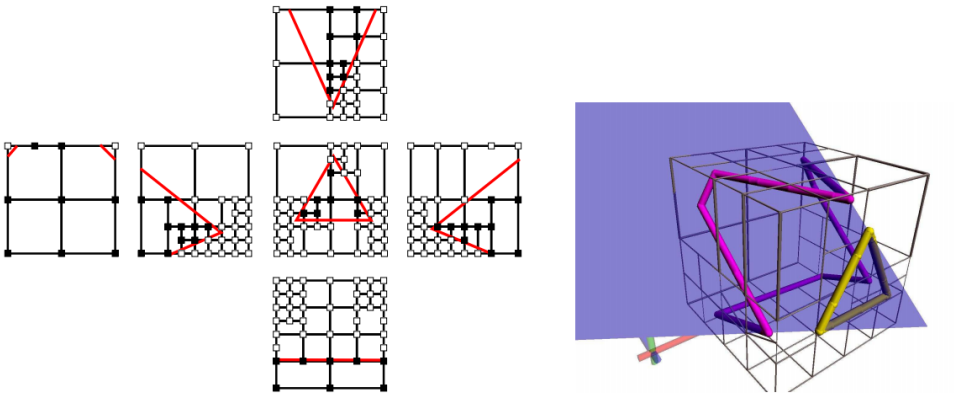

Figure 3.5: *An exploded cube showing the segments created on each of its faces, and the combined cube which depicts the surface passing through it, matching the created components. Image source: Ho* et al. *(2005)*

Extracting Surface from Leaf Cells

When all the face segments are generated, the CMS algorithm proceeds by extracting a surface for each leaf (surface containing) cell. This stage of the algorithm could be decomposed into four subsections, each of which consists of smaller algorithms for achieving their specific purpose. The extraction stage consists of the following steps, tracing of the components, detecting of cell sharp features, resolving internal ambiguity and triangulation.

- Tracing components is done by linking together the segments of the six faces of a cell into a circular 'component', which is a 3D contour approximation of the surface in that cell.
- 3D sharp feature preservation is achieved in CMS by additional sampling and solving an equation by singular value decomposition. Again, the position of the exact sharp feature in the cell is saved, as with the face sharp features.
- Internal disambiguation is related, once again, with the sharp features. If the cell's corner signs allude that there might be a internal ambiguity, the sharp features are checked for intersection with the other components in the cell. If there is an overlap in the volumes of any two components, they are to be classified as joined.
- Triangulation of the cell surface is dependent on the case. If there is no sharp feature, the average of all the vertices in the component is found and is used as the center of a triangle fan. If a component has a sharp feature, the point at the sharp feature serves as the centre of the triangle fan. If there was an internal ambiguity and the volumes of two of the components are signified as joined, then a dynamic programming algorithm is ran, that correctly triangulates the resultant cylindrical surface approximation.

Listing 3.1, gives an overview of the CMS algorithm through the three major steps which where covered.

Listing 3.1: *The overview algorithm of the CMS publication. Algorithm source: Ho* et al. *(2005)*

```
procedure CubicalMarchingSquares(HermiteData H)
        InitializeBaseGrid(B)
        for each cell C in B
                SubdivideCell(H, C)
        end for
        for each leaf face f
                GenerateSegment(f);
        end for
        for each leaf cell C
                ExtractSurface(C)
        end for
end procedure
```

3.4 This Project

As stated earlier the goal of this project was the development of a C++ library for isosurface extraction, which is based on an existing algorithm, which claims to have some of the technical features spoken of, above. The choice for basing the project on the CMS algorithm was made after extensive research in the area and the advice of experts in the field. This project does not implement all of the features of the CMS algorithm as describes by Ho *et al.* (2005). This project does not also implement the strict algorithm, however stays faithful to the main ideas, namely the simplification of Marching Cubes into Marching Squares, and meshing with an adaptive resolution without cracks or the need for crack-patching. It does this, whist maintaining inter-cell independence, and produces manifold polygonal meshes.

Chapter 4

Implementation

This chapter uncoveres the backbone of the technique and lays the foundation of this specific implementation of the CMS algorithm. As this has been a loose implementation of the algorithm, not all parts of the original algorithm are covered. Also there are some sections which have been done in a different way, from the original CMS. The naming convention of the terminology is not followed strictly from the original paper and some new terms are introduced, as necessary. Some of the tables used are modified from the Kokopelli project Keeter (2013), which also uses similar conventions.

4.1 Algorithm Logic, Tables and Structures

Logic

Before laying the foundations of the program it is important that some abstract rules are established and kept consistently throughout the entire algorithm. Such rules include the coordinate system (frame of reference), the dimensions, the sampling order, the indexing of vertices, edges and faces in a cell, as well as their relationships. The tables and figures, shown below, depict the approach taken in this project. They do not resemble the only way of addressing the problem, however, as mentioned

earlier, a 'way' must be agreed upon, and kept consistently throughout. In this project a 'lef-handed' coordinate system is used, as seen in Figure 4.1, which visualises the coordinate system and the ordering adopted throughout this project. The ordering is also explicitly laid out in the ordering table, Table 4.1.

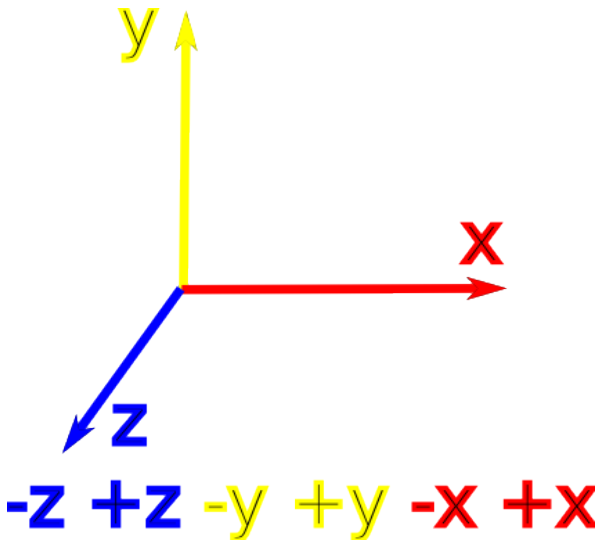

Figure 4.1: *The coordinate system used throughout this project, following the 'left-hand rule' and the ordering of the all 3D operations.*

Table 4.1: *Order Table - a signed table visualising the order in which operations are undertaken.*

X	Y	Z
-	-	-
-	-	+
-	+	-
-	+	+
+	-	-
+	-	+
+	+	-
+	+	+

The order in which faces are visited in each cell, follows the convention established in Figure 4.1. This is the same order as the order of sampling, which the algorithm runs, namely Z, Y and X. Hence the numbering of the faces and verties of the cell are indexed in a likewise manner, Figure 4.2.

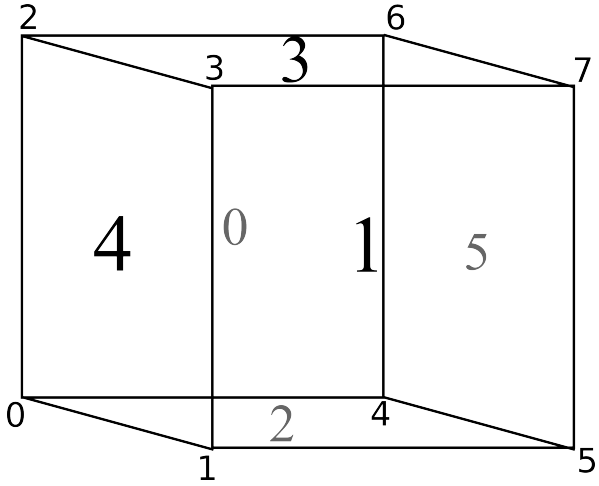

Figure 4.2: *The vertex and face indices in a single cube. The corners vertices are marked with 'Arial' font, whilst the faces are marked with 'Times New Roman'.*

Similarly the cell edges are defined and kept consistent through the whole implementation, Figure 4.3.

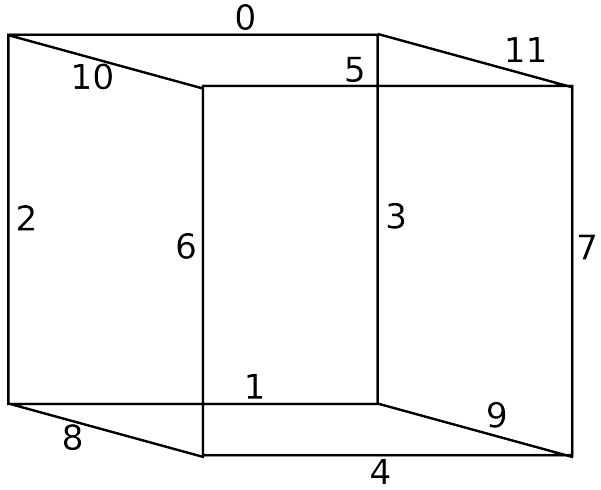

Figure 4.3: *The indexing of the cell edges.*

Because of the nature of the CMS algorithm, each face in a cell is evaluated individually and then all six faces are combined as a cube (cell). Therefore, it is necessary that the *face* vertex and edge order are also predefined. In this project, the configuration is the following, as seen in Figure 4.4. Whereas Figure 4.5 depicts the whole configuration of the way faces are ordered and the relationship between local face edges and vertices, on an exploded cube.

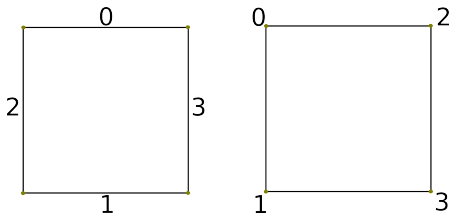

Figure 4.4: *The ordering of the face edges (left) and vertices (right), required for the Marching Squares algorithm.*

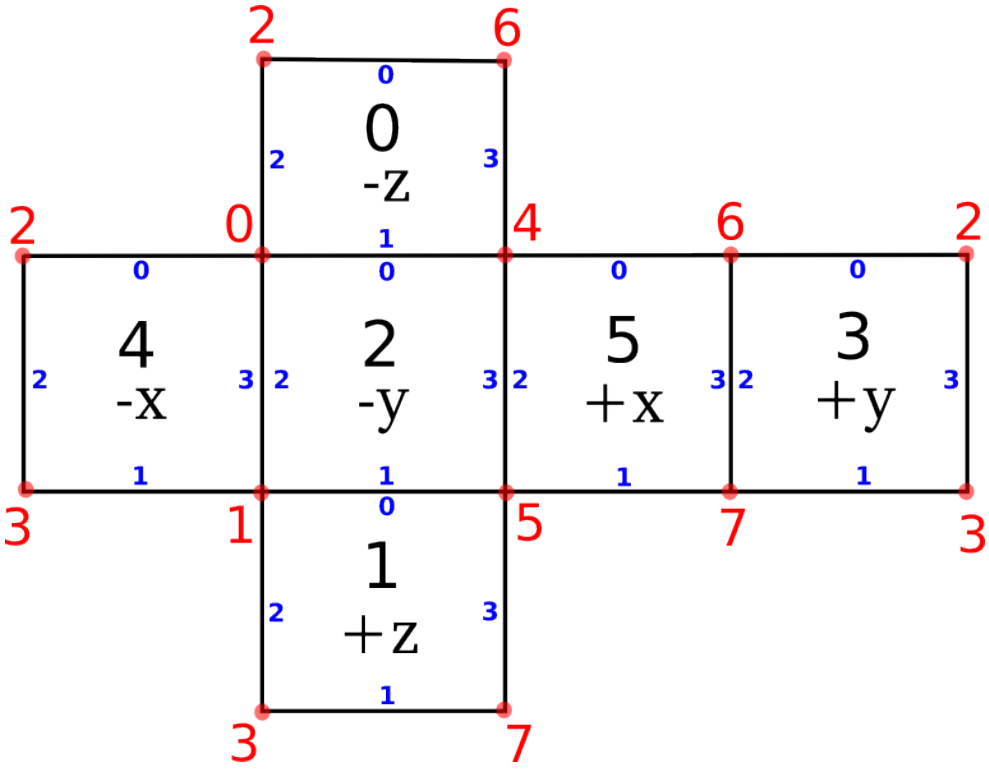

Figure 4.5: *An exploded cube, showcasing the indexing and ordering of the* ***cell faces*** *(black),* ***cell vertices*** *(red), and* ***face edges*** *(blue).*

Tables

In the core of the algorithm only one table is necessary. That is the classic Marching Squares lookup table, which has 16 cases. The 16 cases could be simplified to 4 cases, by using symmetry and mirroring. This 'simplification' has been left out, in order that confusion is avoided, as the face edges are considered when linking the segments. Thus a single uncompressed table was employed, Table 4.2.

The tessellation process is done dynamically and a table is not needed

Table 4.2: *Marching Squares Table*

Case	Strip 0 Edge A	Strip 0 Edge B	Strip 1 Edge A	Strip 1 Edge B	Corners Inside
Case 0	-1	-1	-1	-1	. . . .
Case 1	0	2	-1	-1	. . . 0
Case 2	2	1	-1	-1	. . 1 .
Case 3	0	1	-1	-1	. . 1 0
Case 4	3	0	-1	-1	. 2 . .
Case 5	3	2	-1	-1	. 2 . 0
Case 6	3	0	2	1	. 2 1 .
Case 7	3	1	-1	-1	. 2 1 0
Case 8	1	3	-1	-1	3 . . .
Case 9	1	3	0	2	3 . . 0
Case 10	2	3	-1	-1	3 . 1 .
Case 11	0	3	-1	-1	3 . 1 0
Case 12	1	0	-1	-1	3 2 . .
Case 13	1	2	-1	-1	3 2 . 0
Case 14	2	0	-1	-1	3 2 1 .
Case 15	-1	-1	-1	-1	3 2 1 0

to generate the triangulations of the separate cases, in this project. Such a table is employed for the Marching Cubes triangulation process. It is possible that a similar table be derived for the much simpler Marching Squares algorithm.

The **Next Position** table 4.3 is required for the component tracing stage. This is an important table as it helps the cube 're-construction' process. Given a face and a face edge, the Next Position table, returns the next face and face edge, which are actually the same edge in the combined cube. Figure 4.6, makes the point clear as to how this table works.

This project uses a few other lookup tables, however they are not part of the MS algorithm, but rather are used for ease, clarity and optimisation. They mainly define the relationships between the different data structures used, which would be covered in the Structures section of the chapter. Some of the other more vital tables in the specific implementation can be seen in Appendix A.

Table 4.3: *Next Position Table - segment linkage table. Given a cell face and a face edge it returns the corresponding face and edge, which would be shared in a joined cube.*

Face	Edge 0	Edge 1	Edge 2	Edge 3
Face 0	3, 0	2, 0	4, 0	5, 0
Face 1	2, 1	3, 1	4, 1	5, 1
Face 2	0, 1	1, 0	4, 3	5, 2
Face 3	0, 0	1, 1	5, 3	4, 2
Face 4	0, 2	1, 2	3, 3	2, 2
Face 5	0, 3	1, 3	2, 3	3, 2

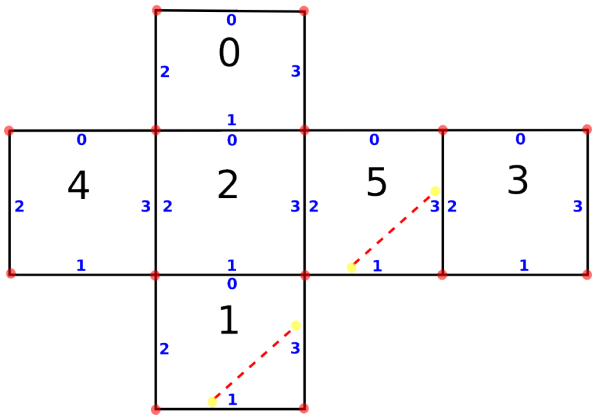

Figure 4.6: *This is an example of how the 'Next Position' table works, showing how just two segments on two faces are being linked together. Using the table, the program will see that the strip on face 1 continues onto face 5 and edge 1, therefore it will merge the vertices on face 1 edge 3 and face 5 edge 1, because they are actually the same vertex.*

Some other parts of the code could be further simplified into look-up tables.

Structures

The main data structures which are used in the algorithm are the two monolythic arrays used to contain all the sample data, and all the edge blocks. Storing such great amounts of memory in a single one-dimensional array is the most efficient and optimal way of storage. For the ease of data manipulation a Array3D class is build as a wrapper, which then accesses the correct elements in the large block of memory underneath.

Some of the other more notable data structures used are, a vertex array and a signed adaptive octree, which holds pointers and indices to the arrays mentioned, linking the data together. The cells of the octree are stored in dynamic memory as their number could be very different depending on the input, and the resolution, due to it's adaptivity.

A **Cell**, Cube, or Voxel, all refer to one and the same thing in this project, and that is an abstract cube in space, which has exact position and dimensions and is part of the Adaptive Octree data-structure. Each cell, except the root, has a parent cell. Each cell could have strictly either zero or eight children. The octree does not support an arbitrary amount of children, as it is not a Branch-on-need Octree (BONO). However, adjacent cells could be on very different levels of subdivision, unlike restricted octrees. The faces between such cells of different subdivision, are called *transitional faces*. Transitional faces are the reason for cracks in adaptive resolution, as seen in Figure 4.7. Figure 4.8 offers a screenshot of a manifold crack-free cube, and a non-manifold cube with cracks. Note that they are not visible, regardless of that the vertices which are not connected between faces, created by different cell, rupture the surface.

The cells containing a surface, are known as *Leaf cells*. Leaf cells, do not have children. Therefore the surface is stored only in the leafs of the octree. The rest of the cells are known as *Branch cells*. Branch cells should have eight children and should not have surface data. As only leaf cells are used for the extraction, the union of the surfaces produced by all the leaf cells would result in the complete polygonised surface.

In this projects the relationship between cells and faces is stored using a half-face data structure, which is similar to the half-edge, however simplified. The data which the half-faces store is enough for the construction of the separate segments and components in all cells, therefore, no other neighbouring information is stored.

Half-face is a data-structure employed for the cell faces and their relationships, both with neighbours, as well as with their parents and children. Each half-face can have four child faces, in which case it is

considered to be a branch face. If a face does not have any children and has a surface, this face is a leaf face, if it has no surface it is considered *empty*. There exists another type of face which is called a *Transitional face*. A transitional face occurs when its twin, or the other half-face on the other side of it, is a different subdivision and they both have surfaces. Transitional faces are a special case and must be dealt with separately.

A simplified half-face structure is required as it will store the information of the transitional faces. It is simplified as, the only additional information that is stored is the address of the 'opposite' face on the adjacent cell, also called it's *twin face*. This is done, to avoid the need for storing a complex array of faces, which have multiple owners. In the case of a transitional face, resulting from different subdivision level on adjacent cells, the half-face structure will allow acquiring the additional *strips* from the cell containing children of 'deeper' subdivision. If those where not taken into account when creating the segments on it's neighbouring cell, the result would be cracks in the mesh at that place where the two cells meet, Figure 4.7.

Strip is a data-structures which contains information about two connected mesh vertices. It is similar to a mesh edge, however it stores additional data, as it holds not only the indices of the two vertices in the global vertex array but also the indices of the local face edges which they occupy, as well as some implementation specific flags. This extra information is required by the component tracing algorithm. There could be a maximum of four strips on a single leaf face. That could happen only in one of the two 'ambiguous' cases and provided that both sides have a face sharp feature. In practice in most cases there is a single strip on each face.

Segments are linked strips, on a face. Where there could be a maximum of two segments on a leaf face and an arbitrary amount on a transitional face. That is so because segment loops can exist on more complex transitional faces, however in practice, this is very rare, and

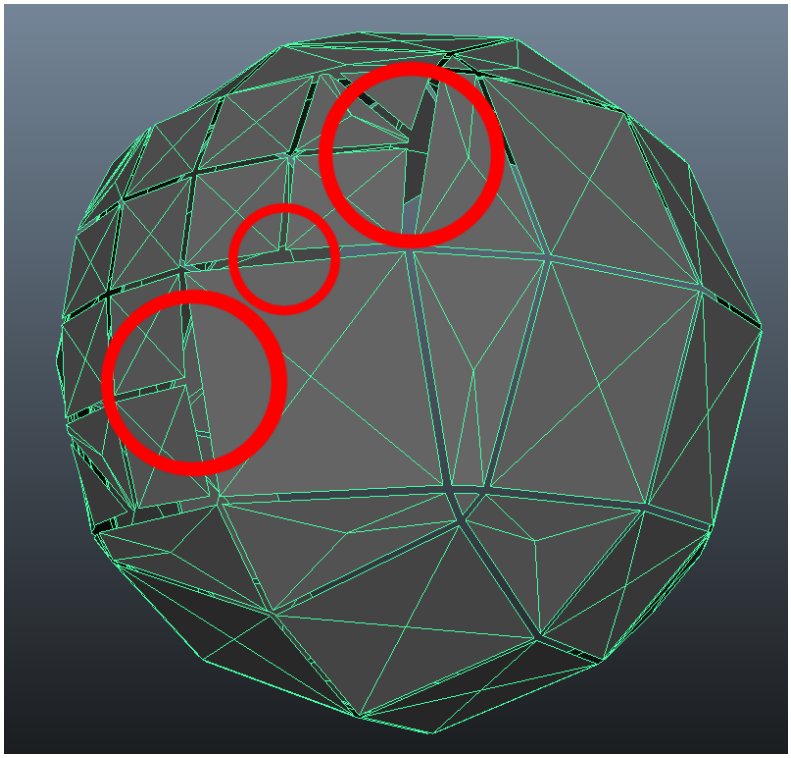

Figure 4.7: *A sphere with gaps and cracks. The cracks are due to inapropriate handling of the transitional faces. The red circles highlight the cracks.*

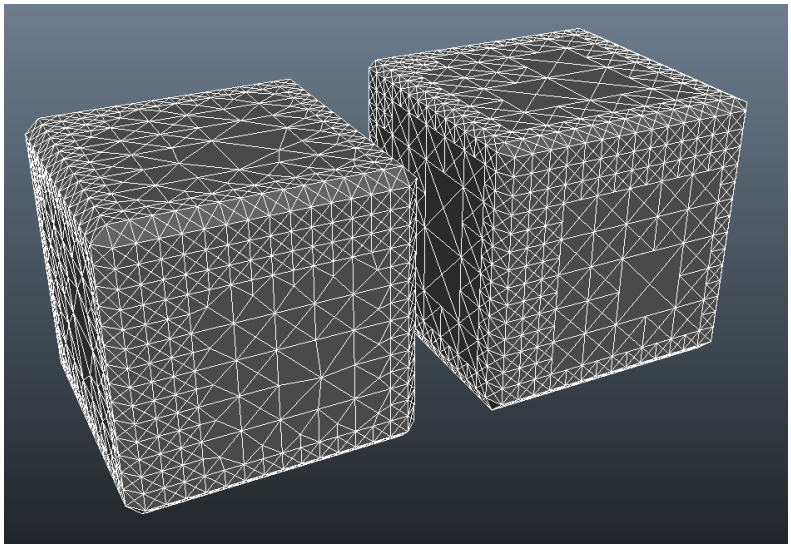

Figure 4.8: *A manifold cube mesh without cracks (left) and a non-manifold cube mesh, with cracks (right).*

more than three segments on a single face are more or less fictional, in the average case. On the other hand, segments consisting of one single

strip could exist, and are perfectly valid.

Components are formed by tracing the segments from each face of a cell and forming loops. Every component must form a loop within a cell, otherwise this would result in topological inconsistency and a non-manifold mesh. In a cell without transitional faces, there could be a maximum of two components. The components in a cell correspond in the most general case to the MC cell configurations. However they can be made up of a significantly coarser piecewise curve if they have had face sharp features and/or transitional faces. Similarly a component could lie on a single face if it forms a loop on that face and does not propagate onto other faces of the cell. This would only be possible on a transitional face, where some part of the surface is passing through that face without coming into contact with its corners, Figure 4.9.

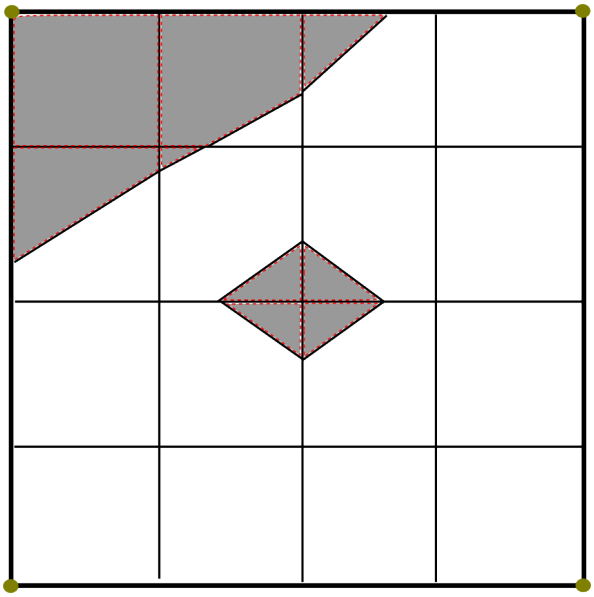

Figure 4.9: *A view of a transitional face and its twin sub-faces. The transitional face has two segments, each made-up of multiple strips, the one in the middle, which closes in on itself is considered a single face component.*

Each component is individually tessellated, however keeping track and registering with the vertices and edges of the global data-structures so that duplicates are avoided. Each tessellated component forms a piece of the final mesh.

4.2 Sampling

The first stage of the extraction of a surface begins with the sampling function. The sampling process is done on a regular grid of the highest resolution of the user specified maximum depth of the octree. As an example if the user input for the extracted mesh is a resolution between 3 and 8, which evaluates to a minimum depth of 2^3 and a maximum of 2^8. This would result in a sampling rate of 2^8.

A different approach is taken to the original CMS algorithm, where all initial input is converted into a standard format, namely Hermite Data, which consists of points and normals at the points. This type of data, first used in the EMC algorithm by Kobbelt *et al.* (2001), serves as the base data on other techniques, such as the DC algorithm, Ju *et al.* (2002). When adaptive sampling techniques are used this data storage is reasonable, however when the surface is sampled on a regular grid, at the finest resolution, the amount of data becomes too great, and a vast majority of it wont be used, therefore it is not practical to sample the function into Hermite Data, unless adaptive sampling is incorporated, which is not the case in this thesis.

The only information that is stored in the initial sampling is just the value of that position in 3D space, which the scalar function, provided by the user, has evaluated. This data is stored into a single one-dimensional array of fixed size:

$$sizeOfArray = (samplesOnX * samplesOnY * samplesOnZ)$$

A 1D array is used over a 3D array for the actual storage of the data, as the sampling process is usually one of the heaviest computations in the algorithm, due to its $O(n^3)$ complexity. Therefore a 1D array is more suitable, as the data is stored in a single continuous block of memory, allowing for faster data access. The 1D array is masked by an Array3D wrapper class for ease of use.

The sampling process, has proven to be one of the more difficult elements in this project, as well as the generation of a correct adaptive

octree on top of it. If done incorrectly it could introduce many small bugs which might have consequences in later stages of the project, causing unwanted behaviour in the program. Extra care should be taken when converting discrete space sizes based on samples, into 3D coordinates from a container of a given size in 3D space.

Note that, both the number of samples and the octree cells have a size which is a power of two. However, there is always one more sample, to ensure that all cells are mapping to the correct samples and that the last cell does not exceed the length of the sample array, as each cell has overlapping samples with it's neighbouring cell. This again could produce problems if not handled properly as the cell which reaches the end of each row will have it's last values out of scope. This is so, because each cell's last sample overlaps with the first sample of its neighbouring cell, which is done to maintain consistent topology, otherwise the octree would become incorrect, resulting in gaps as could be seen in Figure 4.10.

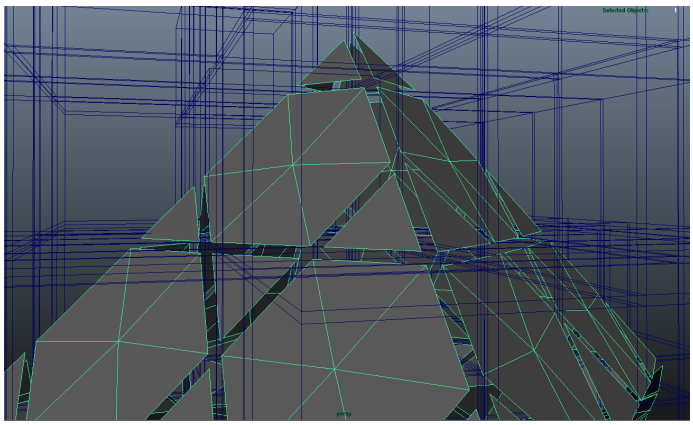

Figure 4.10: *Gaps in the mesh, produced due to a bad implementation of the octree, and cells with non-overlapping samples.*

4.3 Creating the Signed Adaptive Octree

The next stage of the program involves the creation of the signed adaptive octree using the sampled data, which was acquired in the sampling stage. In this thesis, the octree creation procedure is performed in four distinct

stages. They are the following:

- The construction of the octree, via recursive subdivision of the cells.
- The creation and population of the half-face data structure, which is part of the cell face relationships, required for the main algorithm.
- Setting up the parent-child relationships of the cell faces.
- Finding and tagging all the transitional faces in the octree.

This is by no means the most optimal solution for preparing the octree data structure that is needed. The four stages, mentioned above, could be combined and shrunk into fewer more concise functions, paving a way for a more optimal solution. Any such improvements will be addressed as matters of future work. The established bottleneck of the program written for this thesis happens to be in the stage which sets up the half-faces, in the octree creation stage, which can vary between a few milliseconds to a number of hours, depending on the mesh which has to be constructed, and the amount of leaf cells that has to be checked.

Recursive Subdivision

The first step in the construction of the final octree is the actual creation of the cells, through recursive, depth-first subdivision. The algorithm recursively subdivides cells, by testing against *two subdivision conditions*, until there is no need for further subdivision, or the finest level of the Octree is reached.

Condition 1: *Edge ambiguities.*
Cell edges are traversed sample by sample, checking each sample against its neighbouring samples, on that edge, and comparing their signs. If there is more than one sign change on any edge in a cell, that edge is considered as ambiguous and that cell must be subdivided, in order that correct topology is maintained. Since, the finest level of the octree is the same resolution as the sampling grid, cases in which, a subdivision is

required but not permitted, should not occur from this condition. Figure 4.11, demonstrates a the process on a 2D cell.

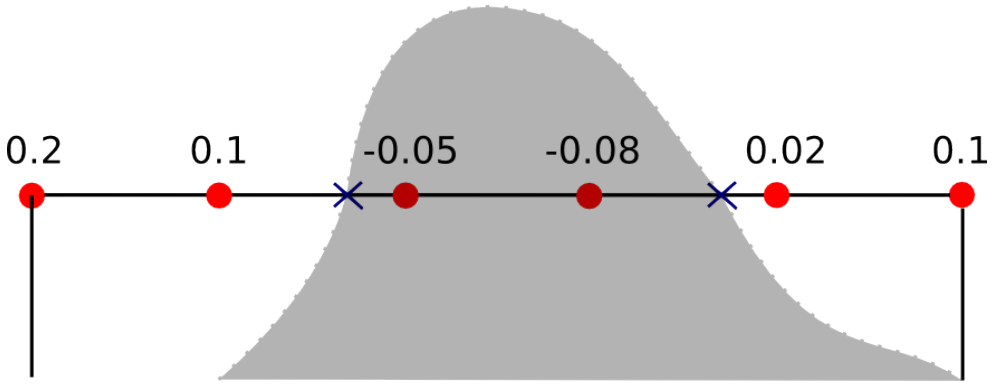

Figure 4.11: *A 2D illustration of a surface passing through a cell edge twice, resulting in a edge ambiguity. The red dots, are sample points and the values are the values that the field function has evaluated at that point in space. Such a cell requires subdivision.*

Condition 2: ***Complex surface.***

The second subdivision condition, checks the normals of the cell corners. The normals themselves are acquired using a forward difference equation, which returns the gradient at the point in space, where the corner is located. This gradient is then normalised so that, the vector might be used as a normal approximation of the isosurface at that point in space. This second check is based on a heuristic, which is defined by the difference in any two of the corner normals in a cell, as shown in Figure 4.12. This is to say, that if the angle between the normals on any of the corners is greater than some predefined threshold, the condition returns true, as there is a potential *complex surface*. Unlike the previous condition this one could demand a subdivision even when the maximum level of the octree is reached, in which case no further subdivision is possible, resulting in loss of precision and therefore a guaranteed non-homeomorphic mesh.

The two conditions complement each other and no further check is required. Care must be taken that the threshold value of the maximum allowed difference between the normals is sensible if the default is not used, as it could be user specified. If the value is too small then the octree would subdivide similar to a regular grid. If the threshold is

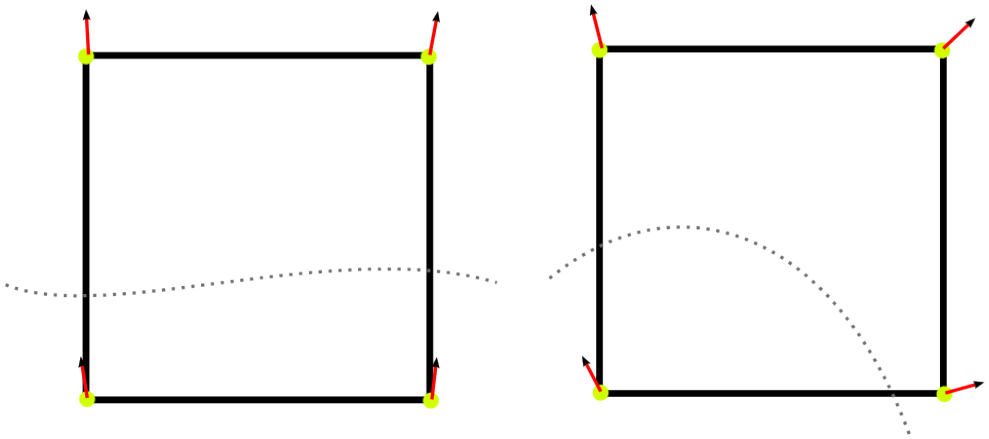

Figure 4.12: *Two cases, scaled down into 2D. In the case of a sensible threshold, one would not register a 'complex surface' (left), and the other would registers a possible 'complex surface', thus that cell would need to be further subdivided.*

too big, some details might be lost as it would not subdivide when a subdivision is required.

Half-face Construction

For the construction of the half-face structure, all cells which are at the same subdivision level should be checked for adjacency. In the case when two cells are *exact neighbours* the appropriate faces are assigned the addresses to their twin face. Where the requirement for exact neighbourhood is that four of the cell corners are shared, as depicted in Figure 4.13.

Establishing Face Parent-Child Relationships

In order to set the parent and children relationships to all half-faces, the tree must be traversed from the root down. The current cell ID resembles the position of that cell, relative to its parent. Because of that fact, the three faces of that cell which are sub-faces to the parent cell, are known based on the cell ID, Figure 4.14 shows the order of the sub-cells. Those three face are all parented to the three parent-cell faces, which they touch. Likewise those faces are added as children to

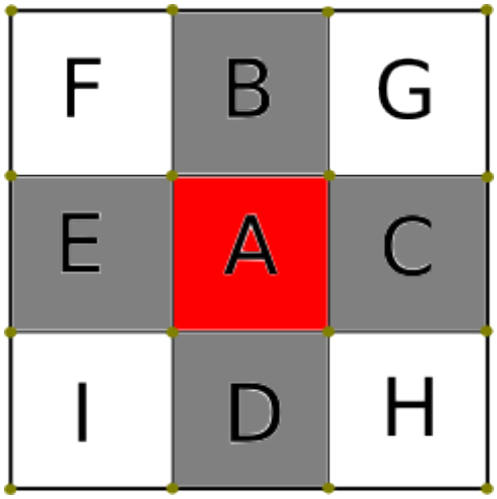

Figure 4.13: *The 'exact neighbours' of cell 'A' in a 2D case. 3D would extend to two more exact neighbours, on the top and bottom of the cell. In this image 'B', 'C', 'D' and 'E' are the exact neighbours of 'A'. Cells 'F', 'G', 'H' and 'I' are not.*

the parent-cell's faces. This process is done recursively for all cells until the whole tree is traversed. At the end of this procedure, a correct parent-child relationship is established for all cell faces.

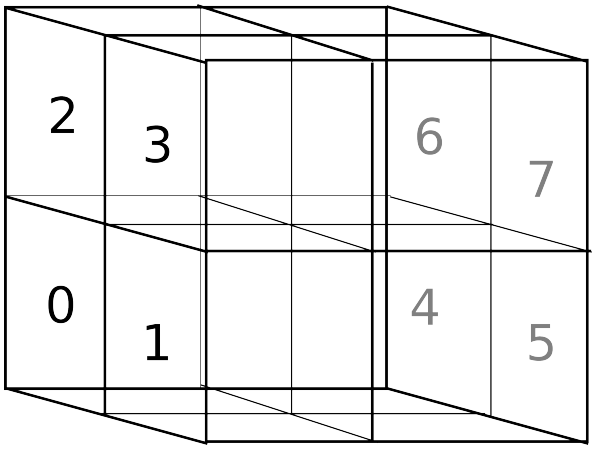

Figure 4.14: *A visualisation of the ordering of sub-cells in a parent cell.*

Finding Transitional Faces

The octree generation concludes with the tagging of the trasitional faces. As it was established earlier, a transitional face is considered to be a

face of a leaf cell which is 'shared' with a branch cell - which has leaf subcells touching the transitional face. A more loose description would be, the face between neighbouring cells of different subdivision, which both straddle the isosurface, Figure 4.15. If the transitional faces are not handled properly, the surface of the mesh would be ruptured, with what are commonly called, 'cracks'. This results in non-watertight meshes. Over the years cracks, in adaptive resolution have been dealt with by different methods, most of which involve a post-process, known as crack-patching. There are numerous methods in which crack-patching could be executed, however it often produces unsatisfactory results. One of the features of the CMS algorithms is that, it meshes at adaptive resolution without cracks, and therefore without the need for crack-patching.

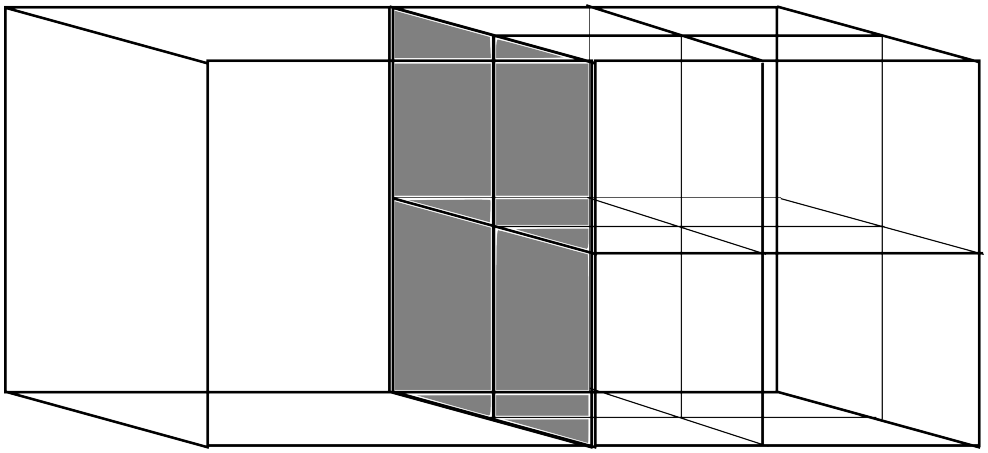

Figure 4.15: *Two cells at different subdivisions. Provided that the cell on the left and the children of the cell on the right are Leaf Cells, the separating face belonging to the larger cell is considered as transitional (grey).*

The location and tagging of the transitional faces is easily done using a brute-force approach. All the octree cells are iterated over, and every cell which is of a 'leaf' state, has to have it's faces checked. The cell faces are iterated and are individually checked for a valid twin face, and if true, the twin face is checked for valid children. If there is a twin and the twin has valid children, this means that the current face should be marked as 'transitional'. This information would be used during the main algorithm so that segments from the twin faces' children are taken into account. This process will ensure that the mesh is 'crack-free' and thus watertight.

4.4 The CMS Algorithm

As previously stated, this project does not follow strictly the steps of the CMS algorithm neither does it implement all its features, however remains loyal to the main contributions of the publication by Ho *et al.* (2005). The main algorithm, as implemented in this project, can be seen in Listing 4.1.

Listing 4.1: *The overview of the Cubical Marching Squares algorithm as implemented in this thesis.*

```
procedure CMS(Octree octree, Mesh mesh)
C <- octree.cells
        for each leafCell C
                for each face f
                        f <- generateSegments()
                end for
        end for
        for each leafCell C
                for each face f
                        if f.isTransitional()
                                f <- resolveTransitFaces()
                        fi
                end for
        end for
        for each leafCell C
                strips[] <- collectSegments()
                components[] <- linkSegments()
                mesh <- triangulateComponents( components[] )
        end for
end procedure
```

Each of the main algorithms' sections would be described separately as they are made up of smaller algorithms, which perform separate tasks.

It is important to mention that the actual mesh vertices are generated during the main algorithm. They are found on every cell edge which has

a sign change, and therefore a crossing point of the isosurface, which is the point in space where the, user provided, field function evaluates to the user-specified isovalue.

Before moving on to the description of the main algorithm, two functions of great importance have to be established, as they are used during the stages of the algorithm to find the points and their normals, which would later become the mesh vertices. Those functions are the *findExactCrossingPoint()* and *findNormal()* functions, which respectively return a positional and a directional vector. They approximate the isosurface position and the normal of the isosurface at that point in space.

Finding the Exact Crossing Point

The zero crossing point is the point in space at which the scalar function returns zero. If the isovalue is zero, then the exact crossing point of the isosurface would be all points in space which evaluate to zero. The isovalue, can be any value, provided that the field function evaluates to such a value in space. The surface value is continuous through space, but isosurface extraction algorithms of the spatial partitioning type, such as CMS, only aim to approximate the surface at reasonable intervals, therefore space is discretised by the sampling process. If two adjacent sample points have opposite signs, this means the isosurface passes somewhere between them. The aim of such algorithms is to approximate, as accurately as possible, this point in space where the isosurface is equal to the isovalue. Therefore, it is necessary to approximate to that point in space, as opposed to just using one of the sample points with opposite signs, Figure 4.16. The approximation is done, by closing in onto the surface, by initially giving only the positions of those two adjacent sample points of opposite signs.

This is a recursive function which progresses towards the isosurface with each next function call. The function takes in two points in space, they are sampled using the original scalar function and if they have an isovalue of opposite signs, the recursive function progresses further.

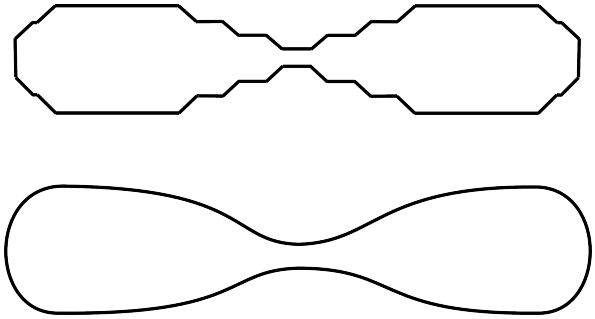

Figure 4.16: *An example of 2D contours, showing the difference between simply picking the closer sample point of two adjacent sample points with opposite signs, and approximating the surface by converging onto it through linear interpolation.*

The progression could either be done through *mid-points*, where the mid point between the two given points is always chosen, or through linear interpolation, which gives a much better approximation, thus finding the surface much faster. Therefore, interpolation is employed for this project. Figure 4.17, provides a visual comparison of the two methods. The *quality* of interpolation could be defined by the user, this value refers to the number of recursive calls which would be made before returning the closest approximation, unless a certain threshold from the isolevel is reached before arriving at the interpolation quality maximum value. Figure 4.18 shows the practical difference between a sphere extracted with interpolation quality of two and a sphere extracted without locating the 'exact' crossing point.

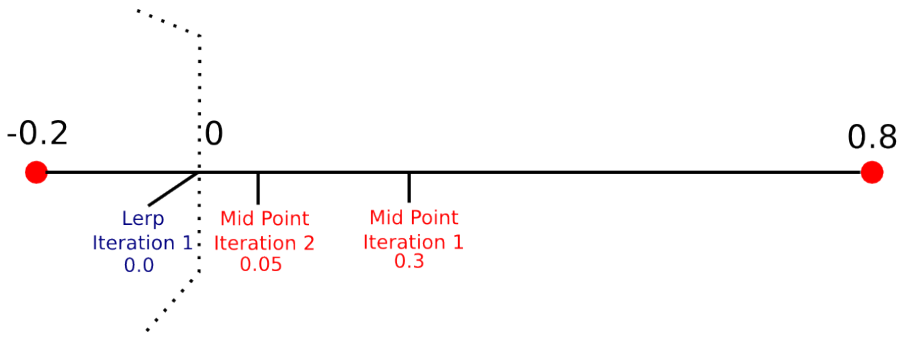

Figure 4.17: *The difference between sinply picking the closer sample point of two adjacent sample points with opposite signs, and approximating the surface by converging onto it using linear interpolation. The mid point method gets to 0.05 after two iterations, whilst the linear interpolation finds the surface after the first.*

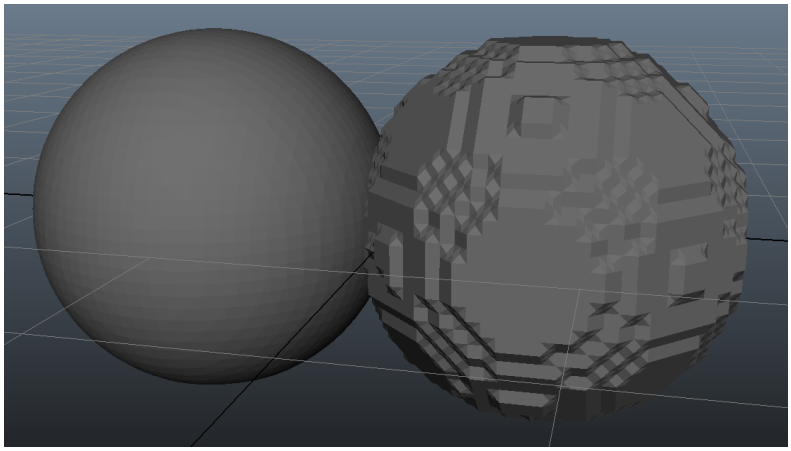

Figure 4.18: *A sphere extracted with an interpolation quality of 2 (left). A sphere extracted without any interpolation, by simply taking the closest of the two sample points (right).*

Surface normals

> *In order for surface normals to be defined along an implicit surface, the function must be continuous and differentiable. That is, at all points along the surface the partial derivatives* $\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}$ *must be continuous and not all zero.*
>
> (Bloomenthal and Wyvill (1997), p.9)

The surface gradient must be found in at any point in space. The forward distance method is used in order that the gradient may be found at an arbitrary sample point in space. The normal of the surface at that point can then be extracted from the gradient. The normals are not only used for exporting a mesh with correct normals, but for other reasons in the algorithm, such as, the construction of the adaptive octree as they are used for one of the subdivision conditions. In the original algorithm they are also used for the disambiguation of the ambiguous face cases and testing against sharp features. The method, used for finding the normals in this thesis is a slightly simplified version of the forward difference, as

it is sufficient for locating the normal approximations, Equation 4.1.

$$\Delta_h[f](xyz) = f(x+h_x) - f(x), f(y+h_y) - f(y), f(z+h_z) - f(z) \quad (4.1)$$

where, h_{xyz} are half of the dimensions of the local bounding box.

Generation of Components

The 'Generation of Components' section can be logically split into smaller procedures, however the two main parts within the generation of components, could be classified as the *generation of segments* and the *linking of segments*. As mentioned earlier a component is made up of segments, segments are made up of strips, and each strip has two vertices. This breakdown is a result of the decomposition of cubes into squares in the main algorithm.

*****Generation of Segments The generation of the face segments is done by recursively traversing the whole octree and for every leaf cell applying the standard MS algorithm on all its faces. Strips are created for every face, depending on its MS case, which is acquired from a lookup table, Table 4.2. Each strip is populated by two vertices, and connects two edges of the face which are taken from the MS table.

The cell edge which should contain a vertex, however, might be made up out of numerous sample points, if the leaf cell is not at the final level of subdivision. Then it is important that the exact two sample points between which the sign change happens, are found and passed to the *findExactCrossingPoint()* function, which was spoken of earlier. Finding them would allow to find the exact sub-edge of the crossing point. This is the edge which should contain a vertex.

Once this edge is found, the edge array is checked for existing vertices on that edge. This is done so that duplicate points are avoided or existing vertices are not overwritten.

- If there is a vertex on that edge, then its index from the vertex array is saved onto the current strip.
- If it is established that there is no previous vertex on that edge, a new vertex can be created and stored on it as well as onto the global array.

Linking of Segments

This part is also referred to as, 'the tracing of components', because when segments are linked together and form a closed loop they form a component. The component is in fact an array of indices so that the triangulation of the vertices is performed in the correct order. And so the triangulated component results in a part of the final mesh.

Resolving Transitional Faces

Before the stage of linking all the segments on every leaf cell, the transitional faces must be taken care of. All the cells are iterated over, from the lowest level of subdivison to the highest. Every cell is checked for transitional faces and if such exist, its strips are modified using its twin face, which would have a much more complex set of strips, as they would have leaf cells of lower subdivision. Once all the transitional faces are updated with the correct strips, the actual linking of segments can begin.

Creating Components

This procedure is done by, once again, looping through all the cells from deepest subdivision up to the root. For every leaf cell, all the strips from its every face are collected and stored into a temporary array. Then a procedure is called which links the strips together, by comparing their vertex indices, and connecting them into a single array of sequential vertex indices. When the first and the last vertices are the same, that means that the component is complete. The function returns this component and it gets stored in the cell to which it belongs. The strips which were used to create it are also respectivelly erased from the temporary cell strip array, and the procedure is repeated while there strips remain in the temporary array. In theory a leaf cell can have an arbitrary amount of strips if it has transitional faces.

Triangulation of Components

The last part of the algorithm, is the actual creation of the polygonal mesh. This stage consists in the triangulation of all the components in every leaf cell. The triangulation process is done based on the number of vertices in a component. A component actually stores only indices to vertices in the vertex array. Since a component, cannot have less than three vertices, the minimum is three. If a component has strictly three vertices then their indices are passed directly to the mesh object, as a single triangle.

Triangle fans

If there are more than three vertices in a component, then the algorithm proceeds by finding the mid point of all the vertices in that component by averaging their positions. A new vertex is created at the mid point, then the component is triangulated using a triangle fan from the mid point and onto every vertex of the component.

Snapping the mid-point to the surface

Since finding the midpoint through the average of all the vertices of the component can result in a point which does not lie on the exact isosurface, an additional step is taken, so that a more accurate position is found for the central vertex of the triangle fan. This step clamps the mid point onto the crossing point of the isosurface using linear interpolation. This step however, is provided as an user option, as it does not always produce more visually appealing results. Figure 4.19, shows two spheres created using the same settings, however one has the 'clamp midpoint to surface' option, set to true and the other, to false.

The Library

The finished program is compiled as a stand-alone C++ library, for isosurface extraction. It provides an intuitive and easy to implement templated interface, which could be implemented as shown in the examples provided with the source code.

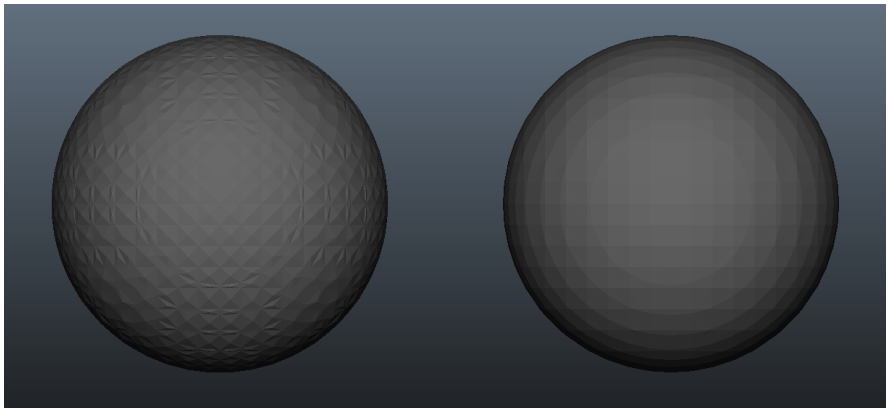

Figure 4.19: *A sphere with mid-points of the traingle fans clamped to the isosurface (left). And a sphere with just averaged triangle fan mid-points (right).*

Exporting Meshes

Meshes are exportable into the standard Wavefront .OBJ file format. The user can export an OBJ of the produced mesh by using the mesh function call to *exportOBJ()*, which takes an export path as a parameter. The program will then automatically export the .OBJ mesh to the specified location once the meshing is complete.

Exporting Octrees

The library also allows for exporting the constructed octree cells, as a python script for Maya. When the script is ran in Maya it creates a cube with the exact dimensions for every cell of the octree, at its precise position. This feature has an option to export only leaf cells or the entire octree. It could be used for visualisation purposes, Figure 4.20, Figure 4.21 and Figure 4.22.

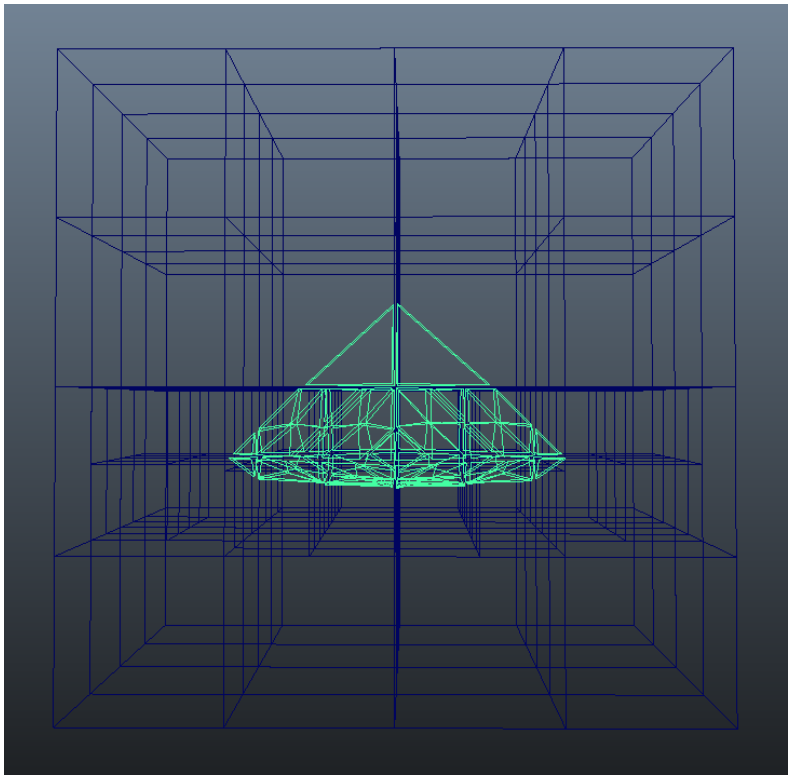

Figure 4.20: *A low-res pyramid mesh and its octree.*

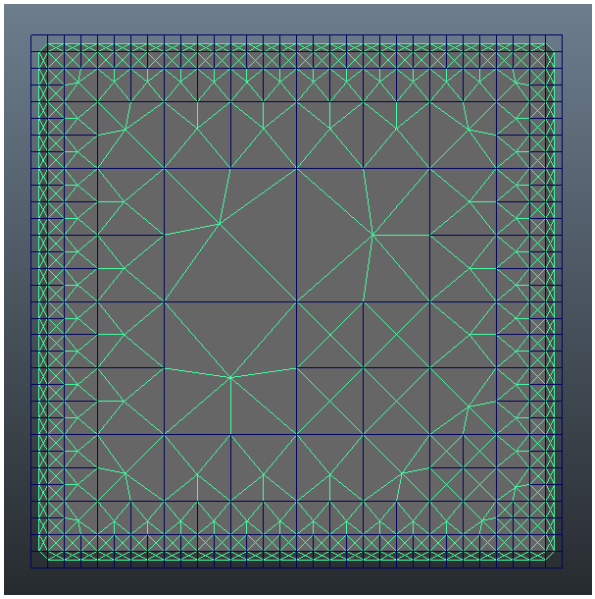

Figure 4.21: *An orthographic view of a cube and the way its topology matches the octree cells.*

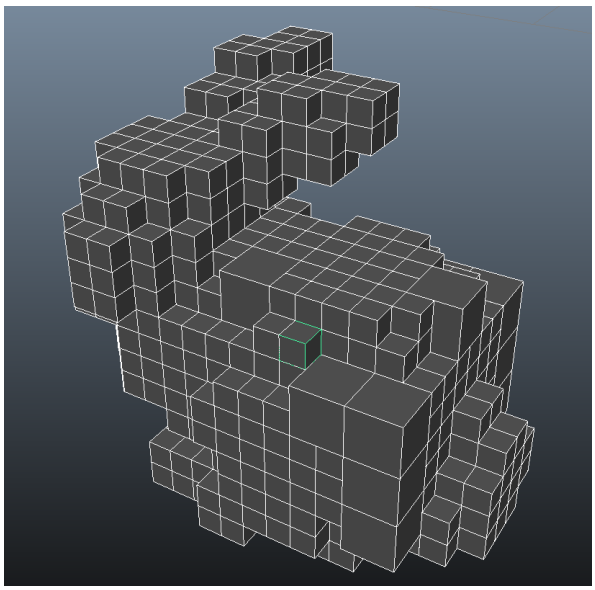

Figure 4.22: *Minecraft Bunny - Only the leaf cells of the octree produced when meshing the 'Stanford Bunny' model.*

Meshing and Exporting Individual Cells

This feature was used as a debugging tool, however it could be useful for visualisation purposes, therefore it remained in the final version of the library. It allows the user to specify desired cells based on their unique ID. The program will then only mesh the surface in the desired cells, and if the octree export is on, it will also only export the desired cells, Figure 4.23.

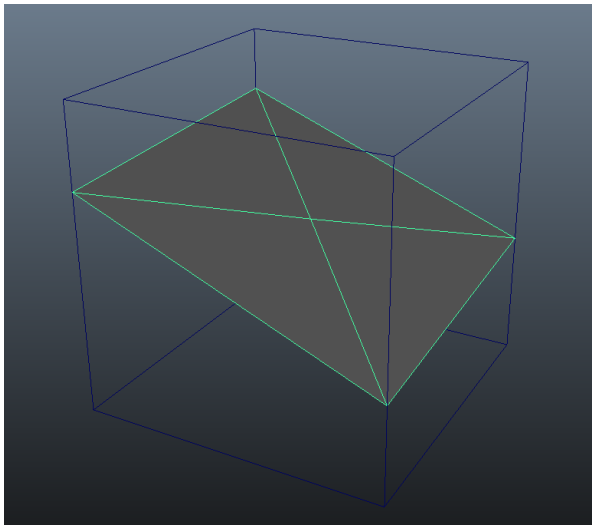

Figure 4.23: *An individually meshed cell, with it's octree cell and mesh surface exported into Maya.*

Chapter 5

Results and Discussion

5.1 Results

This section demonstrates meshes, polygonised with the library created for this project, based on functions of space. Each mesh, has its features and resolution highlighted.

Figure 5.1 depicts a cube meshed from a function. It show how the topology simplifies on in the middle of the cube faces and gets denser as surface detail approaches, which in this case happens to be the corners of the cube.

Figure 5.2 shows two cones meshed from an algebraic function. The difference between them is only the complex surface threshold, which is 0.88 for the left cone and 0.98 for the right cone. The complex surface threshold is the cosine of the angle between the normals and is checked against all pairs of corners in a cell. As it can be seen a slight change in this value can lead to a big change in the resultant mesh.

The three torii in Figure 5.3 are meshed by different isosurface extraction algorithms, at the same resolution. Here adaptive resolution is turned off for the CMS torii by setting both its maximum and minimum levels to the same value. In this case all meshes are produced at a resolution of 32. MC on the left and DC on the right polygonise with a smaller poly-count due to the way they triangulate. The DC torus displays some

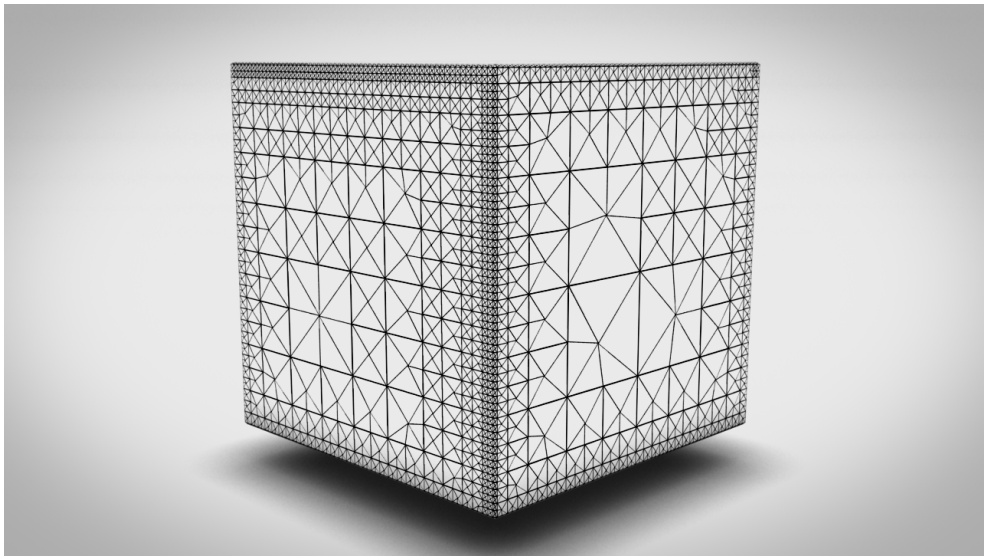

Figure 5.1: *A cube function, Algorithm: CMS, Vertices: 6,222, Triangles: 12,440.*

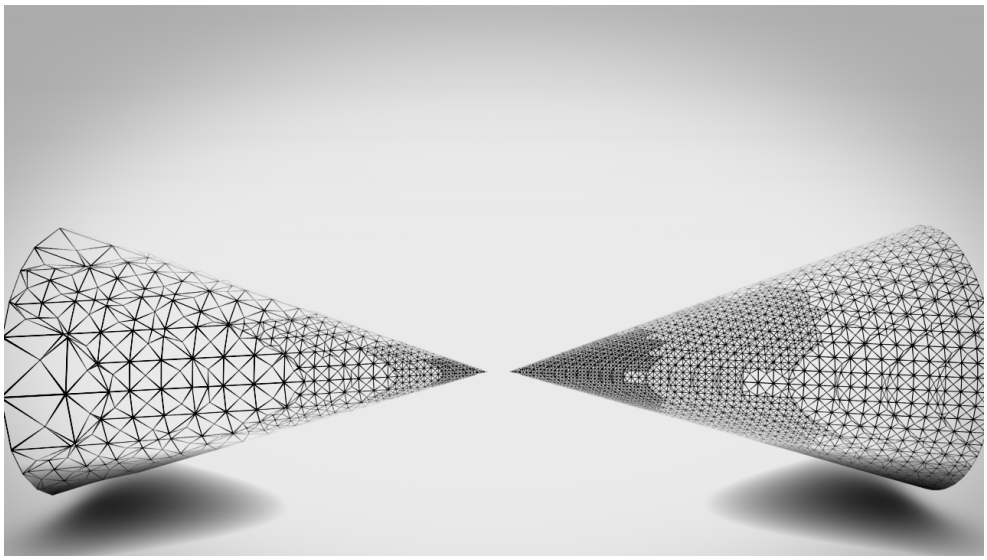

Figure 5.2: *A cone function, Algorithm: CMS. LEFT: Complex Suface Threshold: 0.88, Vertices: 1,865, Triangles: 3,708. RIGHT: Complex Suface Threshold: 0.98, Vertices: 9,183, Triangles: 18,286*

self-intersections in its centre.

The rest of the results are not obtained from algebraic functions but rather from signed distance fields from existing meshes. Thus a comparison with the original mesh could be drawn and retopologisation could be considered. Figure 5.4 shows two models of spanners. The original

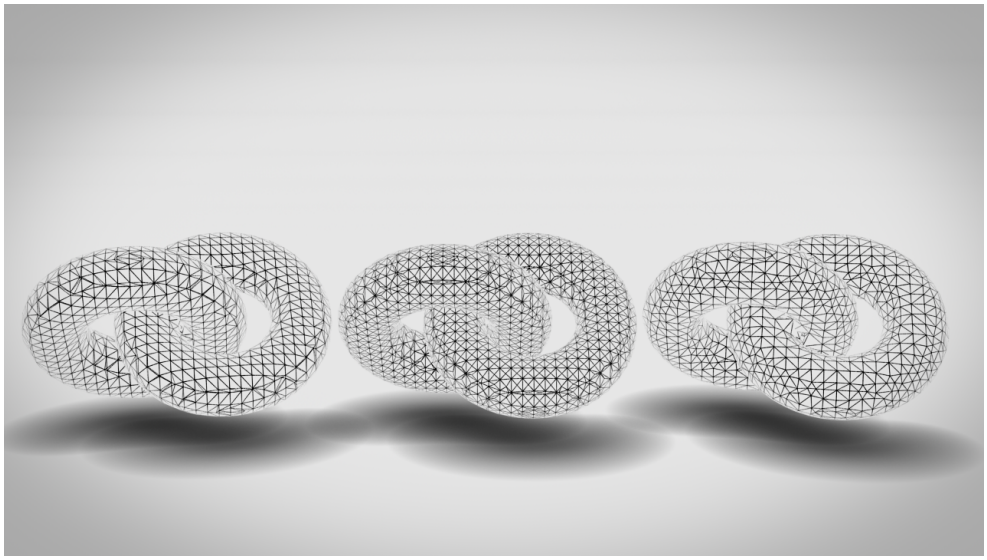

Figure 5.3: *Linked torii function. All Resolutions: 32, Algorithm: MC (left), CMS(centre), DC(right). MC[vertices: 2,224, triangles: 4,464]. CMS[vertices: 3,888, triangles: 7,776]. DC[vertices: 1,888, triangles: 3,840].*

model is to the right of the image, whilst its retopologised version is on the left.

When considering adaptive resolution or sharp features, a CAD/CAM model which is often used is the 'fandisk' model. Figure 5.5, compares the original to the extracted CMS mesh, whilst Figure 5.6 compares the CMS with a MC extraction sampled at the same resolution - 256.

The following images take the same comparison approach, and compare an original model, with two models meshed by this project's CMS and the original MC, both sampling at the same resolution, Figure 5.7, Figure 5.8, Figure 5.9 and Figure 5.10.

Figure 5.11 shows a Stanford bunny model on the right and its extracted version on the left. Detail is lost in this version as the complex surface threshold is set to 0.7, which is too low, to be able to capture all the surface detail.

Figure 5.12 is another extracted mesh based on a model from the Stanford 3D Repository, Stanford (2013). The dragon is meshed between a resolution of 4 and 256. This is a complex model for extraction due to

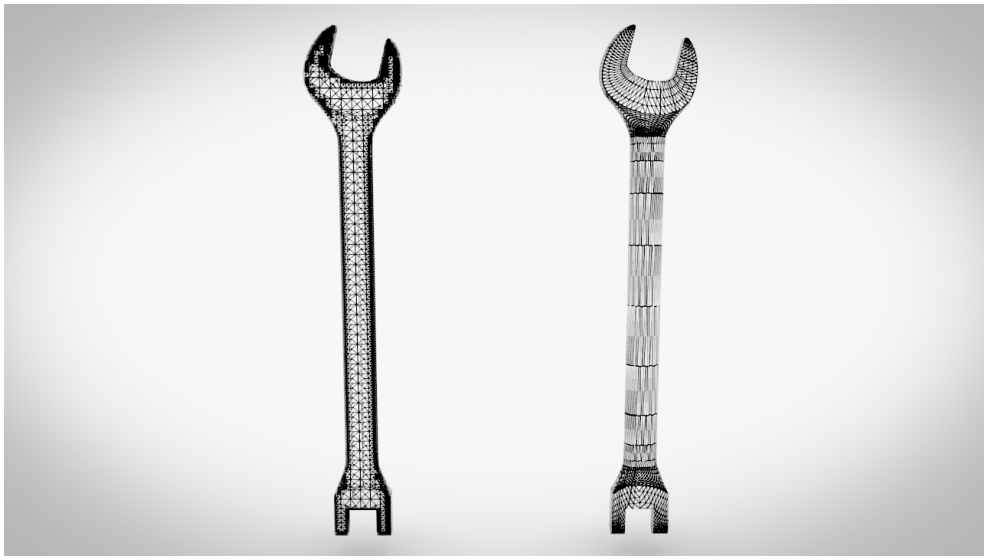

Figure 5.4: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 11,421, Triangles: 22,838] RIGHT[Original Model]. Model modified from: BlendSwap (2014)*

its vast amounts of miniature surface detail and many concavities, the algorithm, does a good job at preserving the detail at a reasonable level, whilst remaining adaptive at a complex surface threshold of 0.85.

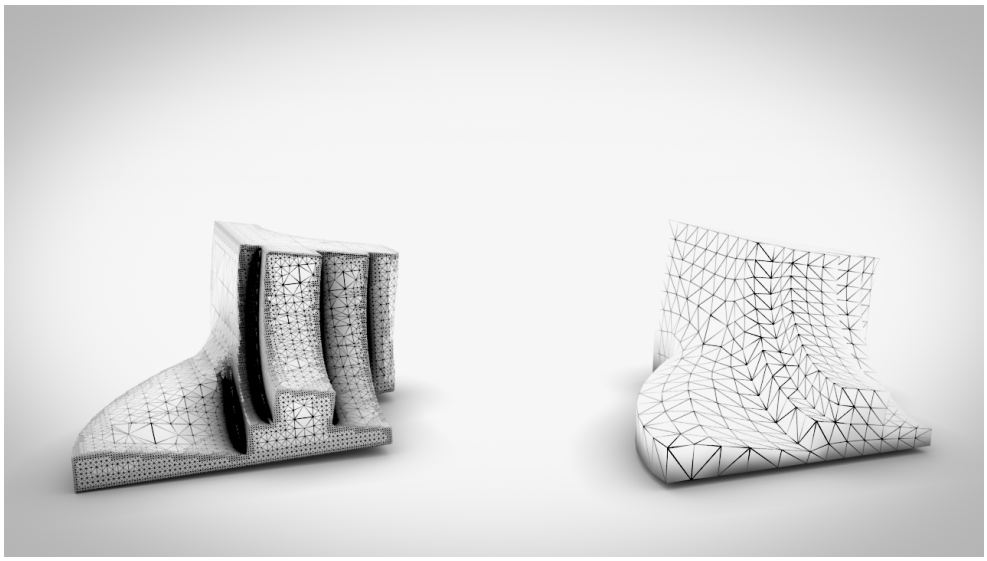

Figure 5.5: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 28,029, Triangles: 56,054] RIGHT[Original model].*

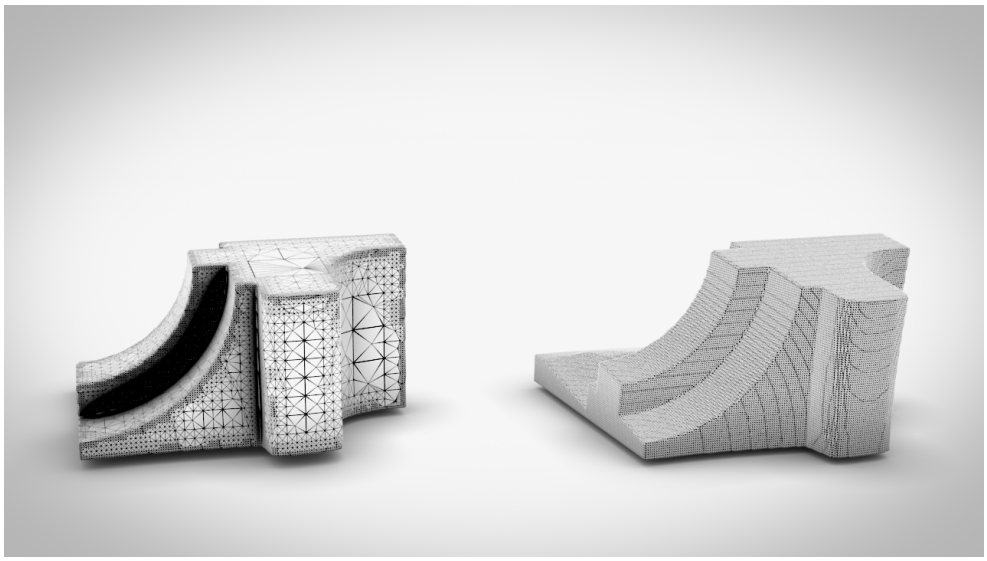

Figure 5.6: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 28,029, Triangles: 56,054] RIGHT[Algorithm: MC, Resolution: 256, Vertices: 86,698, Triangles: 173,428].*

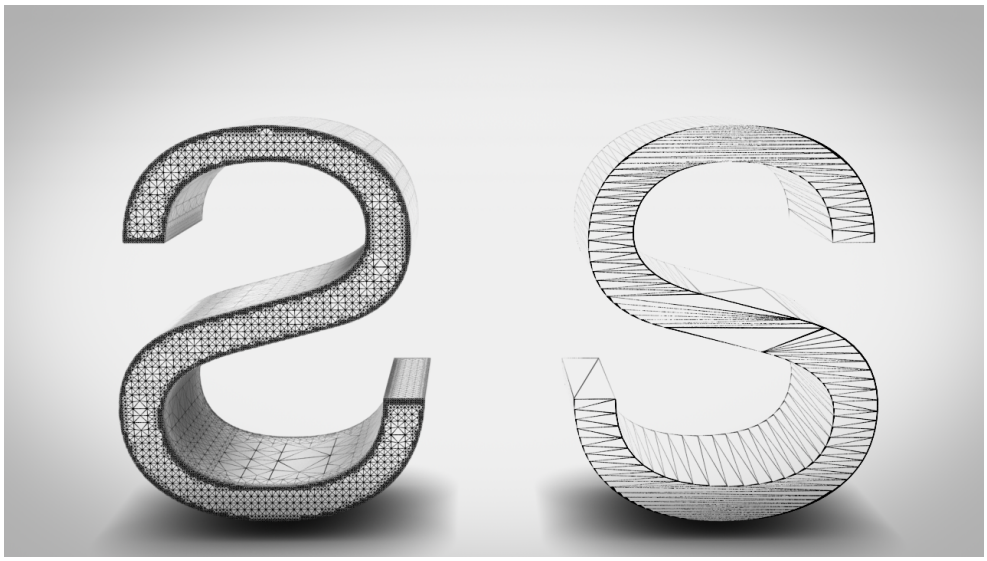

Figure 5.7: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 43,277, Triangles: 86,550], RIGHT[Original Model.]*

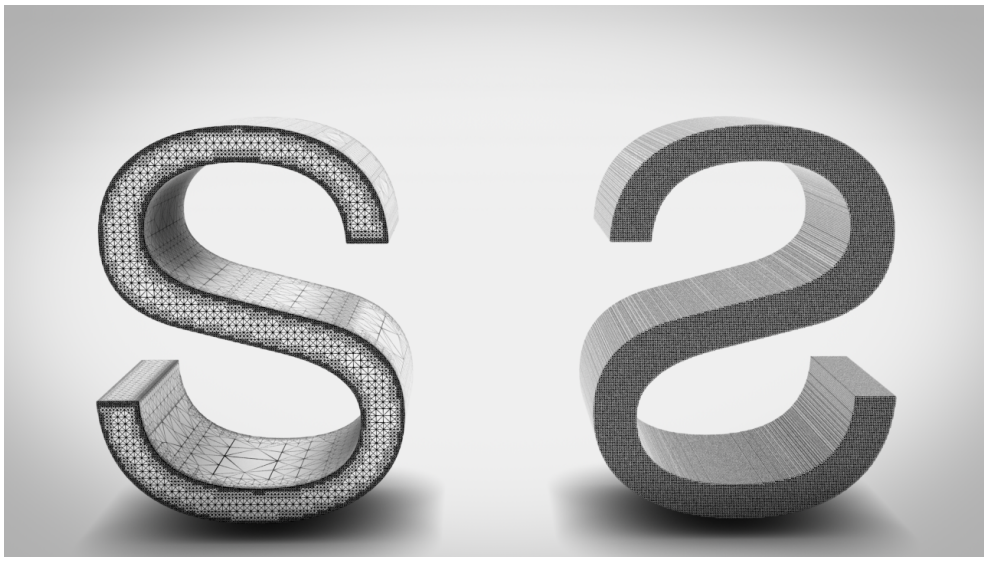

Figure 5.8: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 43,277, Triangles: 86,550], RIGHT[Algorithm: MC, Resolution: 256, Vertices: 167,756, Triangles: 335,508].*

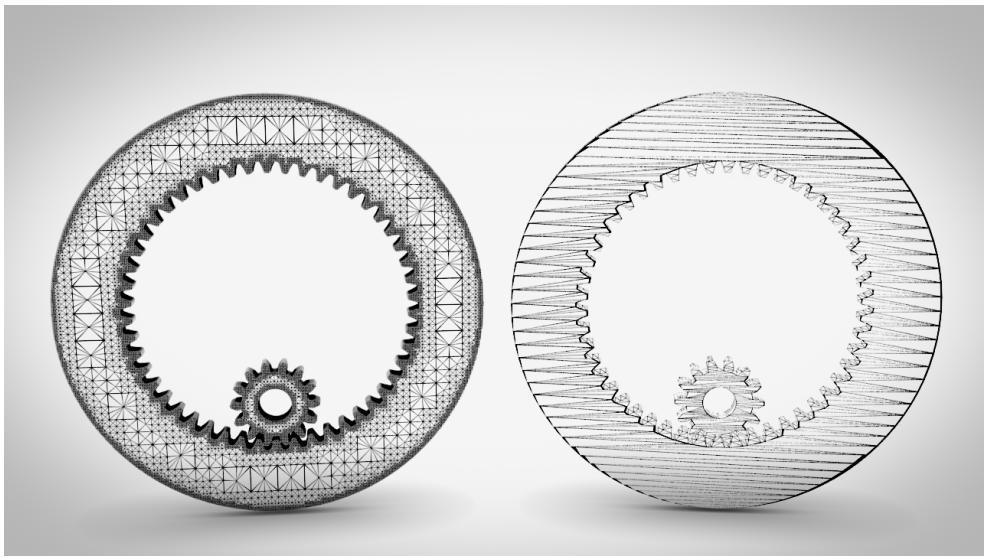

Figure 5.9: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 72,938, Triangles: 145,892] RIGHT[Original Model]. Model modified from: BlendSwap (2011)*

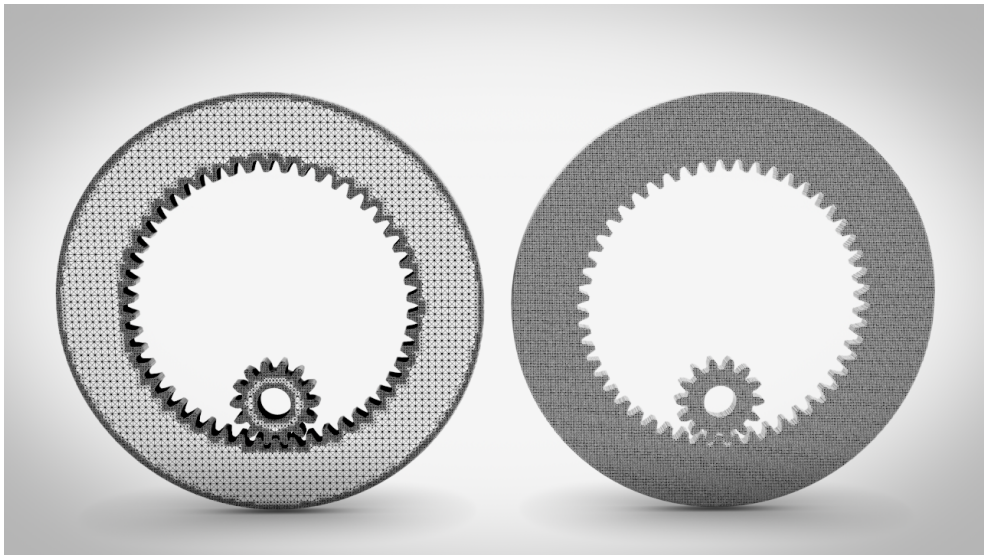

Figure 5.10: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Vertices: 72,938, Triangles: 145,892] RIGHT[Algorithm: MC, Resolution: 256, Vertices: 86,698, Triangles: 173,428]. Model modified from: BlendSwap (2011)*

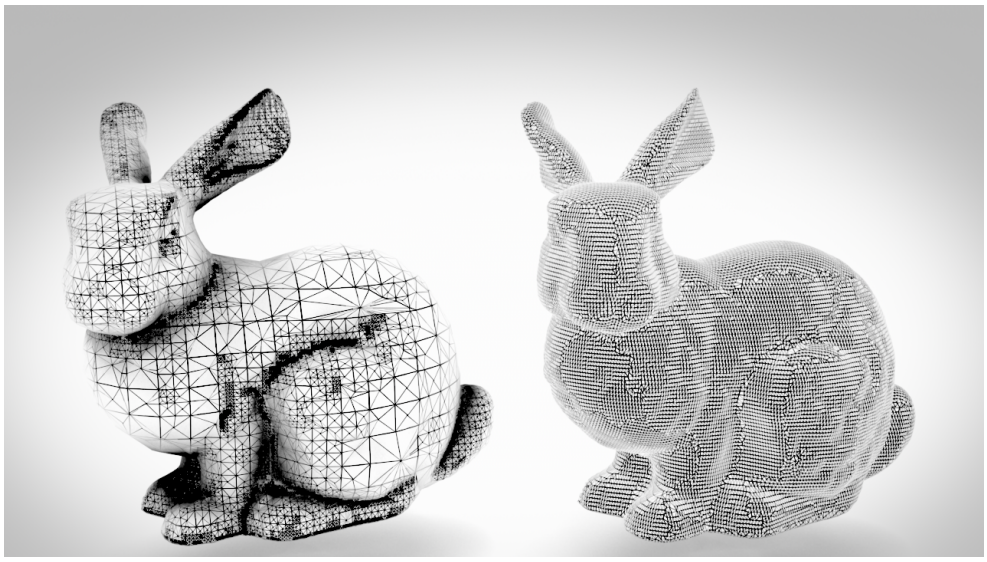

Figure 5.11: *LEFT[Algorithm: CMS, Resolution: 4 - 256, Complex Surface Threshold: 0.7, Vertices: 36,036, Triangles: 72,064], RIGHT[Original Model]. Model courtesy of: Stanford (2013)*

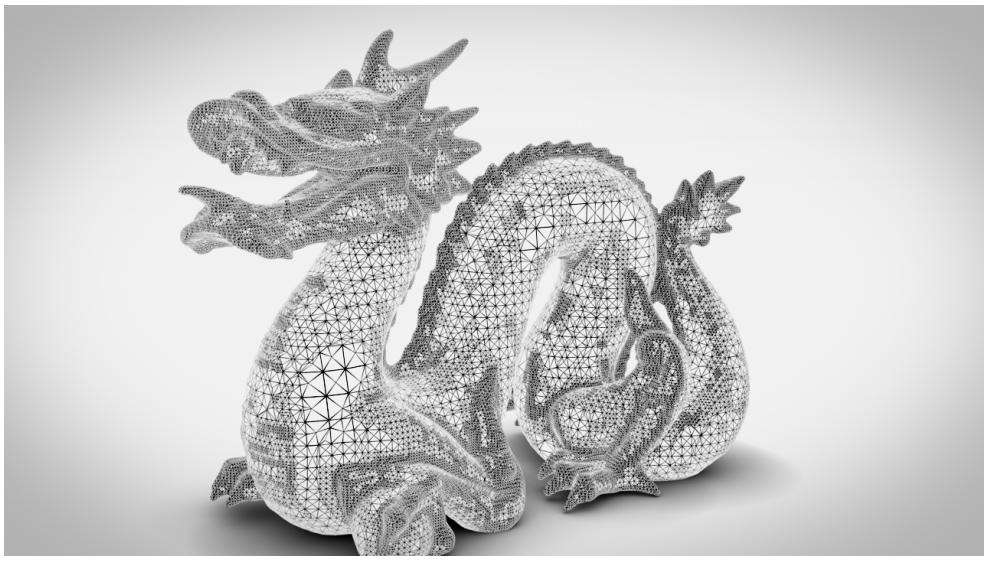

Figure 5.12: *Algorithm: CMS, Resolution: 4-256, Complex Surface Threshold: 0.85, Vertices: 131,054, Triangles: 262,080. Model courtesy of: Stanford (2013)*

5.2 Discussion

From the results depicted above, it is visible that the CMS algorithm is theoretically plausible, even from the loose implementation based on it, which was developed for this thesis. This implementation lacks many of the features which are mentioned in the original algorithm, as preservation of 2D an 3D sharp features and face and internal disambiguation, as they have been beyond the scope of this project. Nevertheless, the current state of the algorithm allows for its expansion in that direction, as those features are based upon the foundation of the algorithm which was successfully developed until the current state.

The comparisons throughout this project were made with the original MC and the DC algorithms. They were mainly aiming to portray the benefit of the adaptive nature of the CMS algorithm, without cracks or crack-patching.

Another thing that draws attention is the usage of the extractor as a retopologisation tool which guarantees manifold results. There is room for improvement in every aspect, however the implementation done for this project proves that there is in fact great potential in the algorithm, despite its complexity. Overall it offers a robust isosurface extractor which could potentially outperform most other algorithms or combinations of algorithms, in terms of mesh quality.

Chapter 6

Conclusion

6.1 Conclusion

Overview

This project aimed to make a partial implementation of the CMS isosurface extraction algorithm. The original algorithm was described in light of previous work in the field and algorithms which are directly related or opposed to the one in focus. The technique implemented in this project was also described in great detail, so that it could be reproduced solely though this document. As this has been a loose and partial implementation of the original algorithm, some of the aspects in the publication by Ho *et al.* (2005) are not identical. Nevertheless, the results achieved match the claims described by the authors of the original algorithm. A comparison was then drawn between this implementation and some other existing techniques, as they were tested on identical input data. This led to a discussion on the plausibility of the algorithm and its potential for expansion, leading to the outcome of this project, which was established to be successful and with great potential.

Outcome

This project resulted in an isosurface extraction C++ library, which can polygonise a user provided function of space.

The function could be specified directly as a field function, or could be provided through some other means. Any scalar field or ADF would be valid input.

The main contributions of the CMS algorithm is that it decomposes a Marching Cubes cell into six Marching Squares faces. Henceforth it is able to achieve all its features as a result of this decomposition. All of its claims are based upon this main idea. This project was based upon the CMS algorithm as it demonstrated potential to be one of the most robust isosurface extraction algorithms to that day. The original CMS claims to have similar speed with MC Lorensen and Cline (1987), preserve sharp features as DC Ju *et al.* (2002), without DC's self-intersections and intercell dependecy, which is the downside of most dual methods. Also it claimed 'cheap' disambiguation, both face and internal, which comes along with the sharp feature preservation, therefore maintaining consistent topology. And most importantly it allowed for constructing meshses of adaptive resolution, without the need for crack-patching. All those functionalities add up to one of the most robust algorithms for polygonising scalar fields even to that day.

This being said, the algorithm, never gained wide spread popularity or acknoledgement by the community. This project set forth the goal to review the claims of the authors of the CMS algorithm, and put its plausibility to the test. The thesis accomplished, what it set forth to do, by a loose and partial implementation of the whole algorithm. By implementing its core features, and proving its claims credible.

Other feature which were spoken of in the original algorithm, which have not fallen in the scope of the project, are the disambiguation procedures and the preservation of both 2D and 3D sharp features. However they are addressed in the Future Work section of this document, as they are possible to achieve with much greater ease, than some the competing

algorithms, and without most of their downsides.

The program created for this thesis, succeeds in providing a loose implementation of the main ideas of the original algorithm, and achieves meshing at adaptive resolution, without crack-patching, resulting in manifold meshes. In addition to that, intercell-independence, is maintained as an outcome, because of the way the algorithm works, which would allow for a potential extension onto the graphics hardware.

6.2 Future work

Additional Features

Since this project resulted in a software library for meshing implicit surfaces. One of the most basic extensions to the work, would be the addition of small features to the library itself. They need not necessarily be an expansions of the algorithm itself, but could be any type of features which will imporve the usability of the library, extend the users options, or improve on the final result. Some suggestions are listed below.

Topological Verification

Etiene *et al.* (2012), present an idea and methodology for topological verification of a mesh extracted from an isosurface. Their methods could 'prove' with high accuracy if a mesh is homeomorphic to the original surface. Furthermore they say that their technique could be implemented as a tool in any isosurface extraction software, and could be used to verify the produced results. The implementation of such a topology verification tool is beyond the scope of this project, however it is an interesting idea and any similar software library could benefit from such a tool.

Mesh Post Processes

Since the output of the program are polygonal meshes, and often such spactial partitioning isosurface extraction algorithms, could produce disproportional triangles, which are either very thin or very small. For this reason the library could benefit from mesh-specific post process func-

tionalities. Some such processes include: *Vertex Wedling* for vertices at close proximity, lying below a certain threshold. *Relaxation of triangles*, sliding the triangle vertices along the isosurface, to achieve more proportional triangles. *Recalculating and exporting the normals.* At the moment the normals at the vertices are being calculated but not exported with the .OBJ file. Also, there are other more accurate ways for computing the normals, than using the Forward Difference method. A number of functions could be provided for the user, such as indexing or analytical normals.

Optimisation

This thesis has focused on the theoretical implementation of the CMS algorithm, thus the optimal software engineering has been overlooked in some parts of the project. Certainly, the improvement of the code quality and the optimisation of some of the algorithms, in terms of datastructures and programming practices, is a matter of future work. However, listed below are some optimisation suggestions on a larger scale.

Adaptvie Sampling

Currently, the program samples the, user provided, scalar field at a uniform grid of the deepest level of the specified octree. For example, if the octree has been declared between level 3 and level 8, the program would sample the function at level 8, which would result in 2^8 or 256 samples. Many of those samples are never considered, thus a much more efficient solution would be the adaptive sampling approach, similar to the ADFs, proposed by Frisken *et al.* (2000). Unfortunately a great part of the program is build upon the assumption of sampling on a regular grid, therefore, such a modification would result in major refactoring of the code.

Online Assignment of Twin Faces

The bottleneck of the current application is located at one of the stages of the octree creation process, namely the assignment of the twin faces or establishing the half-face datastructure. This is so, because it involves

a nested iteration through all the cells, followed by a series of computationally expensive checks, such as exact neighbourhood and others. This stage still remains the bottleneck of the program, and a more adequate refactoring of this stage could lead to a change the total algorithms time complexity by an order of magnitude. The most efficient way to dealing with this problem, would be the online assignment of the twin faces, during the octree construction and recursion. This, however is not straightforward and was therefore left as a matter of future work.

GPU Implementation
Another possible optimisation, is one that the authors of the CMS algorithm, suggest themselves. The benefit of one of the features of the CMS algorithm, namely its *intercell independence* is that it allows for a full implementation of the algorithm onto the GPU.

Improvement and Extensions

The improvements and extensions of this project's library into full implementation of the main algorithm are some of the main ways through which the algorithm could be further improved and upgraded, to handle more isosurface extraction problems.

Sharp Feature preservation Preservation of face sharp features is the next logical thing to be implemented into the library produced for this thesis. Due to the fact that the solution to this problem is based on the backbone of the main algorithm, this further add-on should not be very difficult to achieve. The theory behind it, is checking for a face sharp feature based on the surface normals of surface-straddling faces, projected onto them. The intersection point of the tangents which are defined by the points and their normals, is a potential 2D sharp feature, which is taken into account when constructing the strips for the face.

The original CMS publication, also calims to preserve 3D sharp features by sampling each resultant component, with already preserved face sharp features. This is done "by solving $[\dots n_i \dots]^T p = [\dots n_i s_i \dots]$ by singular value decomposition, where s is the location of a sample point,

n is a sample normal, and p is the location of the sharp feature" Ho *et al.* (2005). If there is a 3D sharp feature, the triangulation is done by a triangle fan similar to the standard method, however the 3D sharp feature position is used as the centre of the fan. This techniques was not covered in this thesis, as it is vaguely touched in the original paper. This feature essentially comes at a relatively low complexity with this algorithm, as it does not require any special approach to achieving 3D sharp feature preservation, as opposed to other specialised techniques, as DC Ju *et al.* (2002).

Internal and Face Disambiguation A problem that must be addressed in order for maintaining homeomorphic topology is the need for disambiguation of the ambiguous cases, both face and internal. In practice, the face ambiguity is much more common for nearly any model, therefore a higher priority should be given to it. It also happens to be much easier to disambiguate than internal ambiguity.

Another feature spoken of in the original CMS paper is the method for *Internal Disambiguation*. As covered in the Technical Background section of this document, Internal Ambiguities are caused by ambiguous configurations of the cell corner signs, when determining whether two components are joined or separated, cannot be established just from the corner signs. The technique for internal disambiguation is based on top of the the extraction of 3D sharp features, as they are used to detect any such ambiguity. They are detected in a similar way to the detection of 2D sharp features. However, in the 3D case, the two components' volumes have to be checked for intersection. If such exists, there is an ambiguity and it could be resolved by meshing the resultant component in a specialised way, which resembles a cylindrical shape, matching the underling surface.

Reduction of Vertices The triangulation process was covered in the implementation section of this document. It was taken from the original CMS paper, however it suffers from one drawback and that is the excessive usage of triangle fans. In this thesis, as in the original paper, if there are more than three vertices which need to be triangulated, this is

done by finding their mid point and performing triangle fan triangulation. This is done in order that the triangulation might be consistent with any sharp features. This however, results in many new vertices, even on faces which do not require that additional vertex, as no sharp feature is found on them. The produced mesh could be thus simplified by applying the triangle fan triangulation only on segments with 2D or 3D sharp features, but if no sharp feature was detected, then triangulate the 'standard' MC way, Figure 6.1. This is a minor detail, which is significantly easier to implement than many of the other proposed future works, however it can make a big difference in the number of vertices on high-resolution meshes.

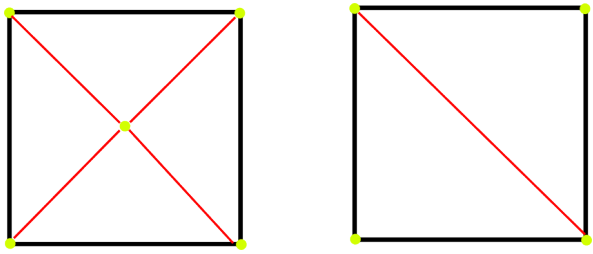

Figure 6.1: *Triangle fan triangulation, with an extra vertex, (left). Simple triangulation as performed in MC, (right).*

ECMS An extension of the algorithm to use an adaptive structure is another interesting future work. Such an extension is first proposed by Ho *et al.* (2006) in the Extended Cubical Marching Squares. Such an extension is based on the contour of the underlying model, and would improve the accuracy, the performance of the algorithm, as well as the mesh quality, 6.2.

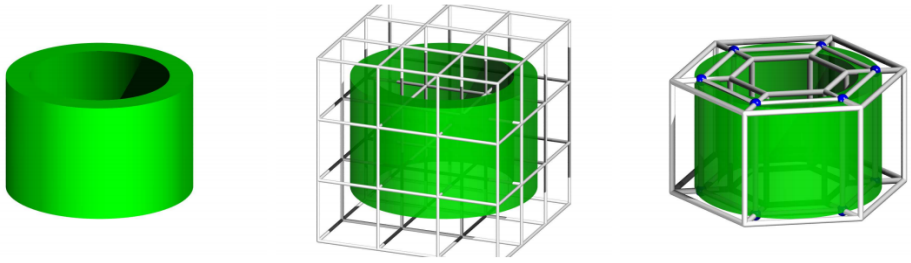

Figure 6.2: *Isosurface extraction on a cuboidal vs adaptive structure. Image sourced: Ho* et al. *(2006)*

Extraction of Normal Maps Barry and Wood (2007) propose a

method for extracting normal maps along with extracting a low-resolution mesh, from the volume data-set. They acquire very convincing results, by simultaneously generating the normal map whilst generating the polygons, 6.3. However, their extraction algorithm of choice is a 'robust Dual Contouring algorithm', their technique does not work as well with Marching Cubes, according to the results from the original thesis based on which the publication was made. A direct extraction of normal maps could be an interesting perspective for visualisation purposes of heavy meshes, with great amounts of small detail. As of this date, this technique has not been tried with the CMS algorithm, and could potentially lead to good results.

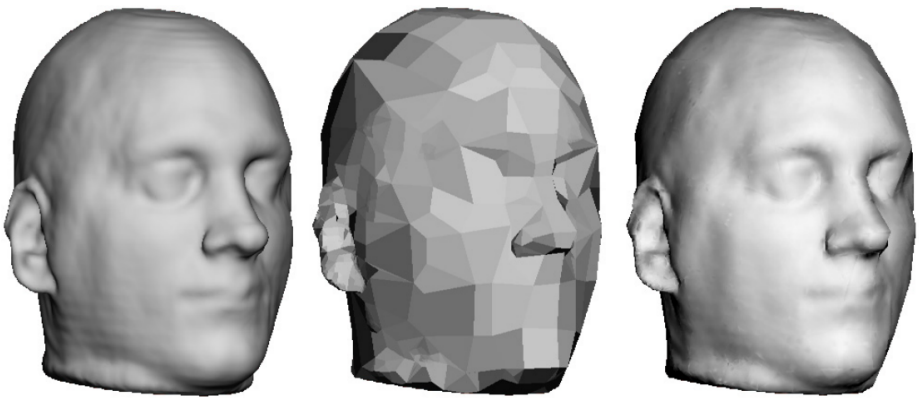

Figure 6.3: *Direct extracting of normal mapped meshes. Meshes extracted from a human head data set. Left: 56,637 quads, Centre: 1,406 quads, Right: 1,406 quads - normal mapped. Image sourced: Barry and Wood (2007)*

Bibliography

1994. The visual computer 11, 1. 52 – 62.

ACM , 2014. Acm digital library @ONLINE. http://dl.acm.org/sig.cfm?id=SP932.

Akkouche S., Galin E. and Centrale E., 2001. Adaptive implicit surface polygonization using marching triangles. *COMPUTER GRAPHICS FORUM*, **20**, 67–80.

Allgower E. L. and Gnutzmann S., October 1991. Simplicial pivoting for mesh generation of implicitly defined surfaces. *Comput. Aided Geom. Des.*, **8**(4), 305–325.

Barry M. and Wood Z., 2007. Direct extraction of normal mapped meshes from volume data. In *Proceedings of the 3rd International Conference on Advances in Visual Computing - Volume Part I*, ISVC'07, Berlin, Heidelberg. Springer-Verlag, 816–826.

Bergen V. D., editor, 2004. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Bhattacharya A. and Wenger R., 2013. Constructing isosurfaces with sharp edges and corners using cube merging. In *Proceedings of the 15th Eurographics Conference on Visualization*, EuroVis '13, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association, 11–20.

BlendSwap C., 2011. Single planetary gear @ONLINE. http://www.blendswap.com/blends/view/14807.

BlendSwap j., 2014. Set of wrenches @ONLINE. http://www.blendswap.com/blends/view/72128.

Bloomenthal J., November 1988. Polygonization of implicit surfaces. *Comput. Aided Geom. Des.*, **5**(4), 341–355.

Bloomenthal J. and Wyvill B., editors, 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Bottino A., Nuij W. and Overveld K. V., 1996. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *In Proc. Implicit Surfaces '96*, 53–72.

CGAL ., 2014. Cgal 4.4 - manual documentation @ONLINE. http://doc.cgal.org/latest/Manual/index.html.

Chernyaev E., 1995. Marching Cubes 33: Construction of topologically correct isosurfaces.

Chu A., Fu C.-W., Hanson A. and Heng P.-A., November 2009. Gl4d: A gpu-based architecture for interactive 4d visualization. *IEEE Transactions on Visualization and Computer Graphics*, **15**(6), 1587–1594.

Crespin B., Guitton P. and Schlick C., 1998. Efficient and accurate tessellation of implicit sweep objects. In *In Constructive Solid Geometry*, 49–63.

Desbrun M., Tsingos N. and paule Gascuel M., 1995. Adaptive sampling of implicit surfaces for interactive modeling and animation. In *Computer Graphics Forum*, 171–185.

Duff T., July 1992. Interval arithmetic recursive subdivision for implicit functions and constructive solid geometry. *SIGGRAPH Comput. Graph.*, **26**(2), 131–138.

Etiene T., Nonato L. G., Scheidegger C., Tienry J., Peters T. J., Pascucci V., Kirby R. M. and Silva C. T., June 2012. Topology verification for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, **18**(6), 952–965.

FountainComputer S. T., 2014. Marching cubes @ONLINE. http://www.fountainware.com/Funware/Mandelbrot3D/MarchingCubes.htm.

Frisken S. F., Perry R. N., Rockwood A. P. and Jones T. R., 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co., 249–254.

Gervaise R. and Richard K., 2002. The marching cubes @ONLINE. http://users.polytech.unice.fr/ lingrand/MarchingCubes/algo.html.

Gibson S. F. F., 1998. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, MICCAI '98, London, UK, UK. Springer-Verlag, 888–898.

Hall M. and Warren J., November 1990. Adaptive polygonalization of implicitly defined surfaces. *IEEE Comput. Graph. Appl.*, **10**(6), 33–42.

Hanrahan P., 1983. Ray tracing algebraic surfaces. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '83, New York, NY, USA. ACM, 83–90.

Hilton A., Stoddart A. J., Illingworth J. and Windeatt T., 1996. Marching triangles: range image fusion for complex object modelling. In *ICIP (2)*, 381–384.

Ho C.-C., Chen B.-Y., Ouhyoung M. and Chen J.-H., 2006. Extended cubical marching squares for surface extraction from various kinds of volumetric structure. In *ACM SIGGRAPH 2006 Research Posters*, SIGGRAPH '06, New York, NY, USA. ACM.

Ho C.-C., Wu F.-C., Chen B.-Y., Chuang Y.-Y. and Ouhyoung M., August 2005. Cubical marching squares: Adaptive feature preserving surface extraction from volume data. volume 24, Dublin, Ireland. pp 537–545.

Ju T., Losasso F., Schaefer S. and Warren J., July 2002. Dual contouring of hermite data. *ACM Trans. Graph.*, **21**(3), 339–346.

Kazhdan M., Klein A., Dalal K. and Hoppe H., 2007. Unconstrained isosurface extraction on arbitrary octrees. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association, 125–133.

Keeter M., 2013. Kokopelli - script-based cad/cam in python / libfab @ONLINE. https://github.com/mkeeter/kokopelli.

Keppel E., January 1975. Approximating complex surfaces by triangulation of contour lines. *IBM J. Res. Dev.*, **19**(1), 2–11.

Kobbelt L. P., Botsch M., Schwanecke U. and Seidel H.-P., 2001. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, New York, NY, USA. ACM, 57–66.

Koide A., Doi A. and Kajioka K., September 1986. Polyhedral approximation approach to molecular orbital graphics. *J. Mol. Graph.*, **4**(3), 149–155.

Levoy M., 2006. Volume rendering @ONLINE. http://graphics.stanford.edu/projects/volume/.

Lewiner T., Lopes H., Vieira A. W. and Tavares G., december 2003. Efficient implementation of marching cubes cases with topological guarantees. *Journal of Graphics Tools*, **8**(2), 1–15.

Lorensen W. E. and Cline H. E., 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, New York, NY, USA. ACM, 163–169.

Montani C., Scateni R. and Scopigno R., 1994. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, **10**(6), 353–355.

Mullner J. M. and Jablokow A. G., 1993. Boundary evaluation using adaptive polygonization. In *Proceedings on the Second ACM Sympo-*

sium on Solid Modeling and Applications, SMA '93, New York, NY, USA. ACM, 481–482.

Newman T. S. and Yi H., 2006. A survey of the marching cubes algorithm. *Computers & Graphics*, **30**(5), 854 – 879.

Nielson G. M., 2004. Dual marching cubes: Primal contouring of dual grids. In *Proceedings of the Conference on Visualization '04*, VIS '04, Washington, DC, USA. IEEE Computer Society, 489–496.

Nielson G. M. and Hamann B., 1991. The asymptotic decider: Resolving the ambiguity in marching cubes. In *Proceedings of the 2Nd Conference on Visualization '91*, VIS '91, Los Alamitos, CA, USA. IEEE Computer Society Press, 83–91.

Ning P. and Bloomenthal J., November 1993. An evaluation of implicit surface tilers. *IEEE Comput. Graph. Appl.*, **13**(6), 33–41.

Ning P. C. and Hesselink L., 1991. Adaptive isosurface generation in a distortion-rate framework. volume 1459, 11–21.

Overveld V. and Wyvill B., 1993. Shrinkwrap: an adaptative algorithm for polygonizingan implicit surface. In *Research Report No. 93/514/19*, 53–72.

Payne B. A. and Toga A. W., September 1990. Medical imaging: Surface mapping brain function on 3d models. *IEEE Comput. Graph. Appl.*, **10**(5), 33–41.

Schaefer S., Ju T. and Warren J., 2007. Manifold dual contouring. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, **13**(3).

Shah N. R., 1995. Homeomorphic meshes in r^3. 25–30. Correct.

Shekhar R., Fayyad E., Yagel R. and Cornhill J. F., 1996. Octree-based decimation of marching cubes surfaces. 335–342.

Shu R., Zhou C. and Kankanhalli M. S., 1995. Adaptive marching cubes. *THE VISUAL COMPUTER*, **11**, 202–217.

Stander B. T. and Hart J. C., 1997. Guaranteeing the topology of an

implicit surface polygonization for interactive modeling. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co., 279–286.

Stanford C. G. L., 2013. The stanford 3d scanning repository @ONLINE. https://graphics.stanford.edu/data/3Dscanrep/.

Szeliski R. and Tonnesen D., 1992. Surface modeling with oriented particle systems. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, New York, NY, USA. ACM, 185–194.

Tsuzuki M. d. S. G., Takase F. K., Garcia M. A. A. S. and Martins T. d. C., 12 2007. Converting CSG models into meshed B-Rep models using euler operators and propagation based marching cubes. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, **29**, 337 – 344.

Upvoid S., 2014. Terrain engine part 2 - volume generation and the csg tree @ONLINE. https://upvoid.com/devblog/2013/07/terrain-engine-part-2-volume-generation-and-the-csg-tree/.

Varadhan G., Krishnan S., Kim Y. J. and Manocha D., 2003. Feature-sensitive subdivision and isosurface reconstruction. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, Washington, DC, USA. IEEE Computer Society, 14–.

Varadhan G., Krishnan S., Sriram T. and Manocha D., 2004. Topology preserving surface extraction using adaptive subdivision. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, New York, NY, USA. ACM, 235–244.

Westermann R., Kobbelt L. and Ertl T., 1999. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, **15**, 100–111.

Wilhelms J. and Van Gelder A., 1990. Topological considerations in isosurface generation extended abstract. In *Proceedings of the 1990*

Workshop on Volume Visualization, VVS '90, New York, NY, USA. ACM, 79–86.

Wilhelms J. and Van Gelder A., July 1992. Octrees for faster isosurface generation. *ACM Trans. Graph.*, **11**(3), 201–227.

Witkin A. P. and Heckbert P. S., 1994. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, New York, NY, USA. ACM, 269–277.

Wyvill G., Kunii T. L. and Shirai Y., April 1986. Space division for ray tracing in csg. *IEEE Comput. Graph. Appl.*, **6**(4), 28–34.

Zhang N., Hong W. and Kaufman A., 2004. Dual contouring with topologypreserving simplification using enhanced cell representation. In *in VIS 04: Proceedings of the conference on Visualization 04*. IEEE Computer Society, 505–512.

Appendix A

Appendix A

Table A.1: *Vertex Map Table - given a face edge, returns it's two corresponding face vertices.*

Face Edge	Vertex A	Vertex B
Edge 0	0	2
Edge 1	3	1
Edge 2	1	0
Edge 3	2	3

Table A.2: *Face Vertex Table - given a cell face and one of it's four verices, get the corresponding cell vertex index.*

Face	Vertex 0	Vertex 1	Vertex 2	Vertex 3
Face 0	2	0	6	4
Face 1	1	3	5	7
Face 2	0	1	4	5
Face 3	6	7	2	3
Face 4	2	3	0	1
Face 5	4	5	6	7

Table A.3: *Face Edge Table - given a face and a face edge this table returns the corresponding cell edge index.*

Face	Edge 0	Edge 1	Edge 2	Edge 3
Face 0	0	1	2	3
Face 1	4	5	6	7
Face 2	1	4	8	9
Face 3	0	5	11	10
Face 4	2	6	10	8
Face 5	3	7	9	11

Table A.4: *Face Edge Table - given a cell edge, returns its two cell vertex indices.*

Edge	Vertex A	Vertex B
Edge 0	2	6
Edge 1	0	4
Edge 2	0	2
Edge 3	4	6
Edge 4	1	5
Edge 5	3	7
Edge 6	1	3
Edge 7	5	7
Edge 8	0	1
Edge 9	4	5
Edge 10	2	3
Edge 11	6	7