uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Jeffrey A. Williams
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Interactive 3D Carving Using a Combined Voxel and Mesh Representation

TITRE DE LA THÈSE / TITLE OF THESIS

Won-Sook Lee
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Amiya Nayak

Doron Nussbaum

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Interactive 3D Carving Using a Combined Voxel and Mesh Representation

Jeffrey A. Williams

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the MSc degree in name of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

Your file Votre référence
ISBN: 978-0-494-48630-6
Our file Notre référence
ISBN: 978-0-494-48630-6

Abstract

This thesis proposes an approach to provide a visually realistic interactive simulation of the effect of removing rigid bounded volumetric portions of a 3D object. The approach processes the volume removal at sufficient rates for realistic real-time rendering, while minimizing the error caused during volume removal operations. We refer to these volume removal operations as *3D carving*. *3D carving* is particularly applicable to the computer simulation of bone-surgery medical procedures performed with a motorized burr tool; however the methods and algorithm presented are generic enough to be used for other purposes such as 3D modeling, destructible objects in 3D games and others.

Our approach represents the volume of the object being carved using voxels while displaying the object to the user using an associated polygonal mesh. We use the *Ball-Pivoting Algorithm* – which has been traditionally used to generate a triangle mesh from a point cloud – to generate the mesh associated with the voxels, but we present a novel extension to the algorithm, the *Dynamic Ball-Pivoting Algorithm*, so that local changes to the voxel set only require local changes to the mesh, whereas the standard algorithm would require a global remeshing. We demonstrate how to apply 3D and 2D textures simultaneously to provide separate external and internal textures for objects that have different skin and internal appearances, which increases the realism of the visualization.

We provide measurements of the performance and accuracy of our approach.

Acknowledgements

I would like to acknowledge the contributions of Dr. Won-Sook Lee to both the writing of this thesis in the form of content, encouragement and support, and in my research in the form of guidance, commitment and inspiration.

I would like to thank my wife Nora for tireless encouragement, inspiration, and confidence. I would also like to thank my immediate family for their help and support, and my employer PMC Project Management Centre for their aid both in financial support and time.

I would like to thank the other student members of Dr. Lee's group for all of their discussion, suggestions, analysis, and friendship during my studies, and in particular Gabriel Telles O'Neill who has expanded upon my work by adding haptic support to the system.

Finally I would like to thank each of the professors I had studied under during my masters classes for their enthusiasm in promoting and advancing the bounds of knowledge.

Table of Contents

List of Figures

List of Tables

Glossary

ρ-ball

Conceptual sphere used by the Ball Pivoting Algorithm (BPA) to determine the next point to add to the mesh given an edge in the front.

BPA

The Ball Pivoting Algorithm, used for generating a triangle mesh that bounds a cloud of points in 3D.

BPA Front

In the Ball Pivoting Algorithm (BPA), the front is the boundary or perimeter of the mesh triangles generated at a point in time when the algorithm is running.

B-spline

A B-spline is a type of spline: a polynomial curve or surface that interpolates between a set of control points in 2D or 3D.

BSP Tree

Binary Space Partitioning tree, a tree used to represent subspaces of a space by hierarchically partitioning the space using planes.

CSG

Computational Solid Geometry, a method of representing objects as a binary tree. Leaf nodes are solid primitives while internal nodes are set operations between the children of the node.

CT Scan

A medical imaging technique in which a 3D representation of tissue density is generated by combining multiple 2D parallel X-Ray measurements around a fixed axis.

DBPA

The Dynamic Ball Pivoting Algorithm, an extension to the BPA proposed in this thesis that allows a local re-meshing of the BPA mesh when the underlying point cloud is modified.

DDF

Discrete Displacement Field, a 3D representation used for carving in which a vector field over a base mesh describes a larger mesh.

DFFD

Discontinuous Free-Form Deformation, a method for deforming a 3D model in an interactive and structured manner and allowing discontinuities (splits) in the model.

FEM

Finite Element Method, a method for solving differential equation problems by breaking up the problem into a discrete mesh that approximates the continuous problem.

GPU

Graphics Processing Unit, a special purpose co-processor designed for efficiently performing 3D rendering usually installed on a graphics card.

Metaballs

A 3D representation in which an object is modeled as an isosurface of a field composed of a number of 3D Gaussian functions.

MRI Scan

A medical imaging technique in which a 3D representation of tissue density is generated by measuring the rotating magnetic fields of hydrogen atoms induced by a stronger external magnetic field.

Polygon Mesh

A representation of a 3D object created by discretizing the objects boundary into a number of adjacent bounded polygonal planes.

RBF

Radial Basis Function, a 3D function whose value depends on the distance from the origin of the function.

Voxels

A 3D representation in which an object is represented as a three dimensional array of regularly sized cuboids, analogous to pixels in a 2D image.

Chapter 1 Introduction

This thesis proposes an approach to provide a visually realistic interactive simulation of the effect of removing rigid bounded volumetric portions of a 3D object at sufficient rates for realistic real-time rendering, while minimizing the error caused during volume removal operations. If one considers the path envelope of a 3D cutting object as it passes through the other object being cut, our approach computes the difference between the shape being cut and the cutting path, and re-computes the shape of the object being cut. For simplicity we refer to the process of removing volume from a 3D object and rendering the result as *carving* in the remainder of the thesis.

Figure 1 shows a 2D example of our carving process.

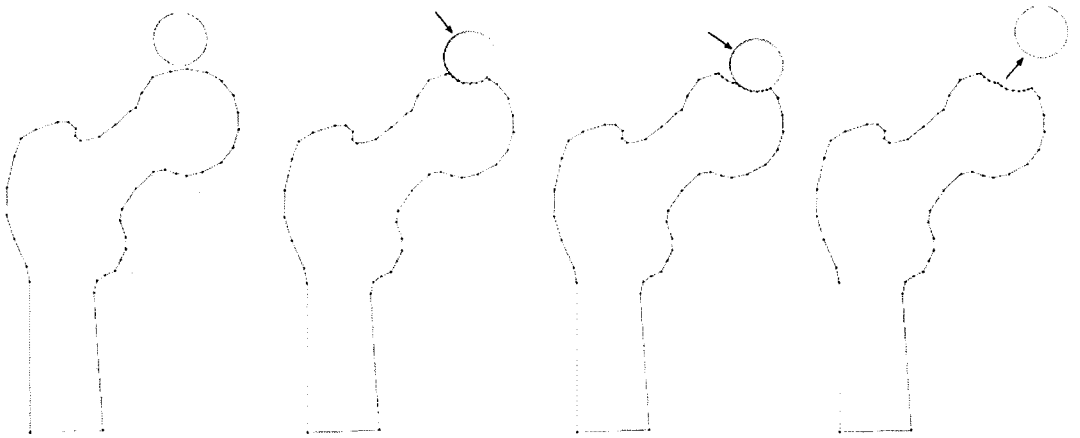


Figure 1: Conceptual diagram of our carving approach in 2D. The top of a femur, represented using line segments as the 2D analogue of 3D faces, is carved by a circle with arrows representing the direction of movement since the last frame. In each frame the area where the circle intersects the femur is removed from the femur in such a manner as to maintain a seamless boundary.

1.1. Motivation

The use of 3D has many applications. A few of the primary applications are described below.

Simulation of medical procedures: Our primary motivation in creating our system is the simulation of certain medical procedures in which excess bone is removed, generally termed *osteotomies*. In some osteotomies, such as the treatment of femoro-acetabular impingement, temporal bone surgery, Hallux Valgus (a disorder resulting in lateral displacement of the first toe), treatment procedures make use of medical burrs and drills. These tools abrade bone surfaces and remove a clearly defined volume of material. To simulate such procedures on a compute system would require representing the bone surface as a 3D object and removing rigid volume from the bone surface at a rate dependent on the shape and roughness of the tool, and representing the external force vector of applied to the tool.

Modeling of 3D objects: Despite the existence of 3D scanning systems and procedural methods, often 3D artists create 3D models by creating and modifying meshes in modeling software. The artists can begin with a previously created mesh or a basic geometric object, and repeatedly subdivide or add faces and vertices, translate, rotate, and scale them, to generate an object.

However, the carving method presented can provide another technique to the repertoire of 3D modelers, allowing them to carve away parts of the object they are modeling similar to real-world carving.

Destructible 3D terrain: Computer games are providing increasing interaction between the player and his or her environment. Volume removal techniques can be used to improve this interactivity for example by simulating simplistic effects of explosions on terrain, the effect of digging a hole in the earth, the effect of biting an apple, of drilling into wood, etc.

1.2. Problem Statement

This thesis addresses the problem of simulating the interactive removal of volume from a 3D object. When addressing such a problem, our requirements are:

Realistic Looking: The effects of the carving should be displayed realistically. When a smooth stroke is used to remove material from the object the result should appear smooth and should not appear to contain visual artifacts. If the object would have internal texture, that texture should appear with the correct orientation and seamlessly joined.

Real-time performance: The effects of carving should appear in real-time. This means that updates to the display should ideally be performed with animation speeds, taking less than 1/24 of a second.

Accuracy: When used for medical simulation purposes, accuracy is very important. The amount of volume removed during simulation should match as closely as possible the amount of volume that would be removed in reality.

1.3. Proposed Solution

Our approach uses two representations of an object: a triangle mesh for visualization and voxels for the carving process. The voxel representation, which represents the volume of the object, is converted to the triangle mesh representation before display. Carving consists of removing voxels from the object voxel set interactively. When voxels are removed the mesh used for display is updated and redisplayed. When generating the new mesh a 3D texture associated with the object is applied to the new faces created by the carving to provide a realistic visualization.

Voxels are used as the volumetric representation since they allow fast computation of the intersection between the carving tool and the surface being cut, they

have been researched extensively for use with haptic models meaning voxel-based carving methods may be integrated with haptic devices with ease, and voxels match medical imaging tomographic image data with high fidelity.

Since the direct visualization of voxels does not look realistic, our method visualizes the surface being cut using a triangle mesh boundary representation. This associated mesh is updated along with the underlying voxel representation when the volume is cut. The boundary representation is computed using a modified version of the classic Ball Pivoting Algorithm (BPA), an algorithm that solves the problem of computing a 3D surface from a set of points (a *point cloud*) [Bernardini 1999]. The BPA begins by with a surface mesh consisting of a seed triangle on the surface of the point cloud that joins three points and the proceeds by repeatedly adding new adjacent triangles to the surface mesh. The new adjacent triangles are formed by pivoting an imaginary ball around an arbitrary edge already on the boundary of the mesh until the ball strikes a new point in the cloud, and creating a new triangle using the edge and the point. Once the mesh has grown to cover the entire cloud the algorithm halts. Though the BPA is intended for use with point clouds, we have shown that it can be applied to voxels with only minor adaptations.

In addition to applying the BPA to voxels, we have also extended the BPA algorithm so that rather than compute the entire boundary representation from the voxel representation at each animation frame when carving, only the part of the voxel volume that is affected by the carving operations and the associated mesh elements are updated. These modifications, which we call the *Dynamic Ball-Pivoting Algorithm* (DBPA), allow interactive frame rates when carving objects.

We choose to use the BPA for meshing and local re-meshing for two reasons: it generates a smooth mesh, and the triangles it generates are nicely-shaped. By nicely shaped we mean they are generally not long and narrow and so are well suited for mesh analysis.

To carve an object, our system requires that the object be represented as voxels. For medical applications this is appropriate since medical images can be easily converted to voxels. However other applications of carving such as modeling and games often use existing 3D meshes. To support these applications our system supports the loading of 3D mesh model files and can convert them to an internal voxel representation.

1.4. Document Organization

The remainder of the document is organized into four chapters. Chapter 2 reviews the state of the art in 3D carving literature. Chapter 3 presents our approach in detail. It explains the classic Ball Pivoting Algorithm that we have extended in our work and the extensions we have made to it, and how we have designed and implemented the system that demonstrates our approach. Chapter 4 presents empirical performance and validation results measured from our system. Chapter 5 concludes and suggests possible avenues of future work.

Chapter 2 State of the Art

This chapter provides an overview of a number of different existing methods for performing 3D carving.

Much of the carving methods from the literature are actually 3D modeling techniques. Though on the surface 3D modeling may seem unrelated to carving, there are a number of modeling techniques that allow the user to carve to model their target object, or include carving as a subset of the technique.

As well, methods for 3D carving are closely related to techniques user to change the shape of a target 3D object using a second 3D object, wherein the target 3D object has all volume within the intersection of the two objects subtracted. There are a number of techniques that achieve this goal known by different names, such as Boolean set operations on 3D objects, cutting, sculpting, and so on.

In the following sections the carving techniques are classified by the representation of the 3D solid on which the techniques operate. For example, implicit and parametric representations treat the object using a set of functions as primitives, while boundary representations represent the object's boundary using a set of planar primitives.

2.1. Carving Objects in Implicit and Parametric Representation

Implicit and parametric modeling is a family of methods for representing solid objects [Bloomenthal 1997]. Rather than explicitly defining the boundary of the solid, implicit modeling represents the solid using an implicit surface, usually using one or more implicit mathematical equations. An implicit equation is one in which specific

values of the input coordinates satisfy an equation. For example, a sphere centred at the origin with radius r can be represented using the implicit equation:

$$x^2+y^2+z^2=r^2 \tag{1}$$

Parametric equations can also be used. These equations represent the surface using an equation for each coordinate, and when used for solid modeling is expressed in terms of two parameters. For example, the following parametric equations express a sphere centred at the origin of radius r in terms of the parameters θ and φ:

$$\begin{aligned} x &= \cos(\theta)\cos(\phi) \\ y &= \sin(\theta)\cos(\phi) \\ z &= \sin(\phi) \end{aligned} \tag{2}$$

In contrast to the above surfaces, an explicit equation might represent one of the variables in terms of the others. For example, an equation for a plane passing through the origin is $z = x+y$. Explicit surfaces have limitations in the shape they can represent; they cannot represent surfaces that fold over themselves or are closed. Implicit surfaces have the benefit that it is easy to determine whether a point is inside or outside the surface; with the function represented as $f(x)=0$ the inside or outside can be defined by $f(x)<0$ or $f(x)>0$.

There are a number of techniques for representing 3D objects using implicit and parametric surfaces.

2.1.1. Constructive Solid Geometry

Constructive Solid Geometry (CSG) represents the object as the combination of a set of solid primitives [Slater 2002]. A primitive can be a parametric equation of a quadric surface (a plane, sphere, cone, cylinder, or paraboloid), or some other simple

regular prism such as a cube. Such primitives are usually chosen since it is easy to compute their intersection with a ray. These primitives form the leaves of a binary tree, in which internal nodes represent rigid transformations (translations, rotations, or scales) of the children nodes, or represent regularized Boolean set operations (union, intersection, or difference) on the left or right sub-tree.

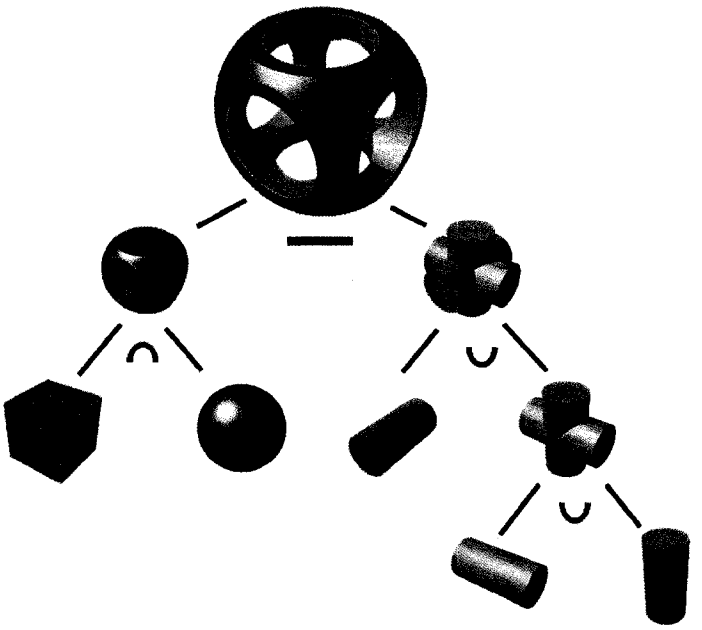


Figure 2: Constructive Solid Geometry (CSG) representation. The object at the root of the tree is composed of intersection, union, and difference operations on descendant nodes [Wikipedia 2005].

Performing carving using CSG is simply a matter of representing the solid to be cut as a CSG sub-tree whose parent is a difference operation with a second sub-tree which is the union of all the positions of the carving tool. However there are two major drawbacks of this carving approach. The first is that the solid to be cut must first be converted to a CSG representation, which is difficult and non-automated for complex objects such as human bones. Second, the CSG representation does not store an explicit representation of the object boundary, and so a ray-casting or boundary representation

must be generated from the CSG before rendering. Despite recent research, ray-casting still does not achieve real-time rates without specialized hardware support. These rendering techniques for level-set methods are described in the section 'Rendering Implicit Surfaces' below.

One solution to these problems is to use Volumetric Constructive Solid Geometry (VCSG) to represent the solid, which allows the use of voxel sets as primitives [Fang 1998]. This removes the need to convert the input solid to a collection of regular CSG primitives but rather to a voxel set. Rendering VCSG structures for display can be done quickly using direct volume rendering techniques [Liao 2002]. However these rendered representations suffers from visual artifacts that cause the visualization to appear blocky and so are not ideal.

2.1.2. Level-Set Methods

Level-set methods represent the 3D solid one or more implicit functions. The boundary of the solid is taken to be an implicit surface, which is the level-sets of the implicit functions; that is, the set of all values in the function domain that yield a certain constant range value, as shown in equation (3).

$$\{(x, y, z) \mid f(x, y, z) = c\} \tag{3}$$

The level-set of a 3D implicit function is also known as an isosurface. Commonly the level-set value c is taken to be zero, without loss of generality. There are a few different methods for representing an object using level-set methods that vary in the manner in which the equations are used to represent the object.

Early methods attempted to fit a single algebraic curve to the surface of the object [Keren 1998][Taubin 1993]. These curves are difficult to compute, do not scale well to huge numbers of points, and are difficult to adapt to changing topology through the translation, addition, or removal of points.

2.1.3. Radial Basis Functions

More advanced methods attempt to blend a number of separate equations. In [Turk 2002] the authors blend *Radial Basis Functions* (RBF) – functions whose value increases as the distance from the point increases – to represent a general surface from a set of known points. The RBF are of the form $b(\vec{x}) = |\vec{x}|^3$, where $\vec{x}$ is a vector. The implicit surface is defined as a linear combination of the basis functions:

$$f(\vec{x}) = \sum_{j=1}^{k} w_j b\left(\vec{x} - \vec{c}_j\right) + P(\vec{x}) \tag{4}$$

In this formulation a number of basis functions are centred at arbitrary points $\vec{c}_j$, weighted with some weight w_j with a first degree polynomial $P(\vec{x})$ accounting for the linear and constant portions of $f(\vec{x})$. It has the interesting property that it has a generally smooth surface as it interpolates the points, similar to thin-plate interpolation. This equation is constrained by forcing the function to interpolate a set of scalar constraints h_i:

$$h_i = f(\vec{c}_i) \tag{5}$$

These constraints h_j are basically the value that the implicit function should take at coordinate $\vec{c}_j$. Since this method is attempting to find the level set of the function at zero, these constraints should be zero for points on the surface, greater than zero for

points inside the surface (internal constraints), and less than zero for points outside (external constraints).

The constraints form a linear system with one equation for each constraint point, plus the internal or external constraint points. The system can then be solved using standard linear system methods.

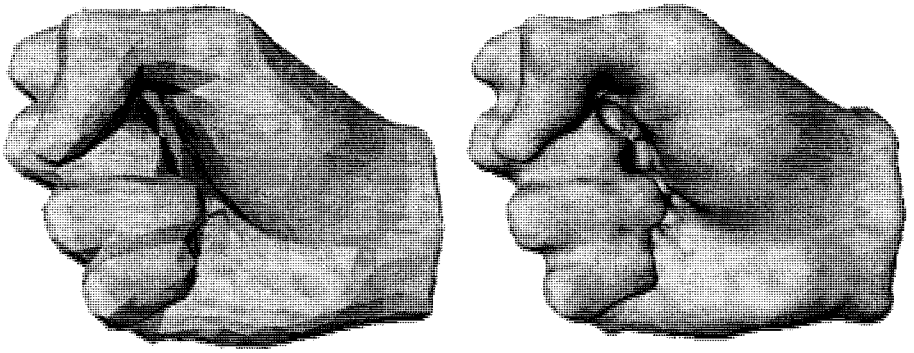


Figure 3: Representing an object using radial basis functions [Turk 2002].

This system could be used to perform carving by removing or translating the input point set. However, each change would require the linear system of equations to be resolved which is slow for large numbers of points, and so is difficult to perform in real-time. The results can also appear unrealistic for smaller numbers of points, as shown in the figure above. The removal of volume is not as accurate with other methods, since the edges of the removed area will appear more bulged, though this might be mitigated with larger numbers of points.

2.1.4. Metaballs

Closely related to the radial basis function method is the idea of *blobby* or *metaball* surfaces [Turk 2002]. Again the surface of the object is represented by blending a number of separate equations. The equations are centred at different points on the

object and represent density functions around each point. The function used is ideally an inverse exponential in the distance from the point, but often Gaussian functions are used to avoid the calculation of a square root:

$$D(x,y,z) = \sum_i b_i \exp(-a_i r_i^2) \tag{6}$$

In equation (6) $D(x,y,z)$ is the density at a given point, which is the sum of a Gaussian function based on r, the distance of the given point (x,y,z) from the centre of ball i. The parameters a_i and b_i for each separate metaball control the radius and elasticity of the metaball. Densities at a given threshold are taken as the surface. Two metaballs are shown in Figure 4.

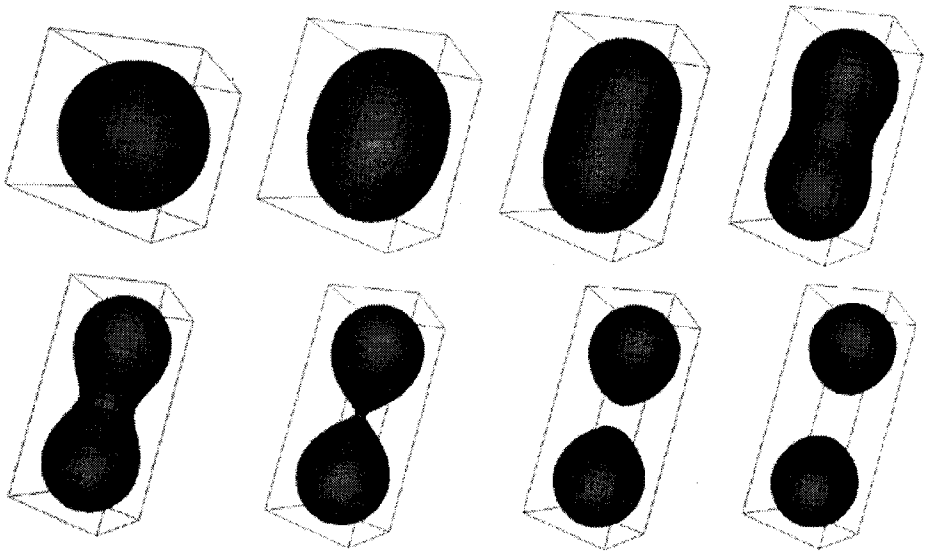


Figure 4: Representing volume using metaballs. As the two metaballs get closer they blend together. [Bourke 1997]

Carving with metaballs can be achieved by removing metaballs from the objects volumetric representation, or through the addition of negative-density metaballs.

However the method still suffers from inaccuracy of cuts since the edges of the removed area will appear more bulged than the actual cut.

2.1.5. B-Splines

In [Raviv 1999] and [Hua 2004] the authors blend trivariate uniform B-Spline functions to represent the surface. These functions have the form:

$$q(u,v,w) = \sum_{i=0}^{l-1}\sum_{j=0}^{m-1}\sum_{k=0}^{n-1} P_{ijk} b_{i,s}(u) b_{j,t}(v) b_{k,u}(w) \tag{7}$$

where $b_{i,s}(u)b_{j,t}(v)b_{k,u}(w)$ are the B-Spline basis functions corresponding to the control point i,j,k. The s, t, and u values control the degree of the basis functions, and thus the continuity of the B-Spline joins (C^{s-2}). The P_{ijk} are scalar coefficients in a volumetric mesh (or control volume) of size l by m by n. Sculpting is done by modifying the scalar coefficient values in the mesh so that the values are inside or outside the object (higher or lower than the level-set value). In [Raviv 1999] the volume is one large function q of the form above with the coefficients initialized to some standard value, so that the initial solid is one large block. The block is then sculpted to the desired shape. In [Hua 2004] the authors extend the work so that existing discrete solids (such as existing meshes or point clouds) can be used as input to the system from which an initial implicit solid is created.

Like with the method discussed previously due to Turk and O'Brien [Turk 2002], these functions are constrained such that the function must be zero at each point on the input point set, and positive for nearby points inside the surface, and negative for nearby points outside the surface. The system of equations is then solved using least squares

methods to generate an initial implicit solid where the object surface is the level-set at zero. This method is fast and provides good results.

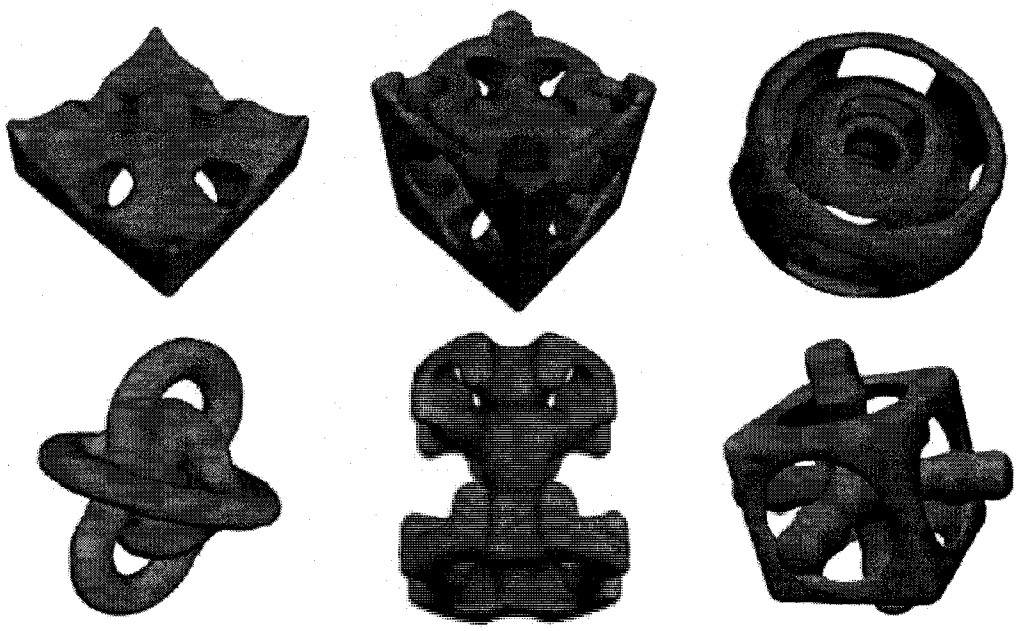


Figure 5: Carving objects using Trivariate B-spline functions [Hua 2004].

2.1.6. Rendering Implicit Surfaces

Generally the efficient rendering of 3D solids is performed from a polygonal mesh boundary representation. This is due to the fact that the computer hardware (3D graphics cards) that renders 3D objects expects this format as input. Since implicit surfaces are a different representation altogether, for visualization they must generally be either converted to a boundary representation amenable to the graphics hardware, or rendered explicitly using ray-tracing techniques.

Ray tracing or ray casting techniques are well suited to implicit surfaces. Ray tracing techniques render a scene of a fixed number of pixels by calculating the ray that would pass through the centre of the pixel into the scene from the person viewing the

scene. Where the ray intersects the earliest surface in the scene the color of the pixel is calculated based on the lighting, color, and material at that point. When rendering implicit surfaces it is relatively simple to compute the intersections of the ray and the surface. Given an implicit equation $f(x) = 0$ and a parametric equation for the ray $p(t) = \vec{p}_0 + t\vec{p}_r$, substituting the parametric equation into the implicit yields a composite single variable equation in t: $f(p(t)) = 0$. The roots of this equation are the t values that generate the intersection points. The problem is then reduced to finding the roots of this composite equation in a stable manner [Hart 1996].

Recently there has been much research into fast ray-tracing techniques. In [Loop 2006] the authors present a method that performs fast ray tracing for algebraic implicit surfaces that are represented by s set of Bezier tetrahedral, which are piecewise smooth. Much of the processing of the method is performed on the GPU. For each tetrahedron, each viewing ray is intersected with the tetrahedral to compute a line segment of the form $p(t) = (1-t)\vec{f} + t\vec{b}$ where $\vec{f}$ is the front point intersected and $\vec{b}$ is the back. Like in the standard ray tracing process, the line segment equations are substituted into the implicit function $g(t) = f(p(t))$ and the roots are found. In order to solve for $g(t)$ and to provide numerical stability in the root finding process $g(t)$ is derived using the blossom, which is computed using a symmetric tensor.

In [Seland 2007] the authors present a method that performs fast ray tracing for algebraic implicit surfaces. Much of the processing is performed on the video card GPU. First a spanning volume is defined that encompasses the algebraic shape to be rendered. This can be any convex hull, but the authors use a polygonal sphere. When rendering each ray is intersected with the sphere to generate a parametric line segment between the

first (front) intersection with the spanning volume and the second (back). These segments can be expressed in the form $p(t) = (1-t)\vec{f} + t\vec{b}$, where $\vec{f}$ is the front intersection and $\vec{b}$ is the second. Like in the standard ray tracing process, the line segment equations are substituted into the implicit function $g(t) = f(p(t))$ and the roots are found. In order to solve for *g(t)* and to provide numerical stability in the root finding process *f(x)* is represented using Barycentric coordinates, and converted to the Bernstein-Bézier basis. Once in this form, a technique called *blossoming* is used to derive *g(t)*.

Since implicit surfaces must be converted to a representation suitable for rendering before being displayed, in order to achieve real-time rates when simulating carving, any changes to the implicit surface – through its primitives in the case of CSG, or its basis functions in the case of level-set methods – must also result in a reconstruction of its renderable representation in less than 1/24 of a second.

2.2. Carving Objects in Boundary Representation

Boundary representations use only the boundary of the 3D object to represent it. A thin shell is rendered to depict the object's surface. The most common boundary representation is a polygon mesh, in which the object's surface is approximated by a set of non-overlapping planar faces such that the each edge of every face is adjacent to an edge of another face, creating a closed mesh.

The earliest methods developed which can be used for 3D carving directly on boundary representations are implementations of Boolean set operations on 3D objects. These operations are like CSG operations, however the primitives in this case are meshes rather than parametric equations. These operations are commonly implemented using Binary Space Partitioning (BSP) Trees [Thibault 1987][Naylor 1990]. It is possible to

define Boolean set operations between two BSP trees (that is, union, intersection, and difference) by calculating the intersections between the planes represented by each tree. Using BSP tree methods it is possible to represent both the object to be carved and carving tool as BSP trees and compute the difference between them at each time interval. However these methods are difficult to implement reliably in the presence of round-off errors [Rossignac 1999]. Since carving is an iterative process where small amounts of volume are repeatedly removed any errors introduced at one carving step will cause further errors later on.

In [Sela 2004] the authors demonstrate a method that operates on the polygon mesh boundary of the shape, and uses a Discontinuous Free-Form Deformation (DFFD) to represent the effect of the cutting blade, deforming the mesh when incising. Extra polygons are added to the mesh under the incision before deforming to help preserve the mesh shape. However this method is more applicable to simulating incision rather than carving since no actual volume is removed during a cut.

Figure 6: Cutting objects in boundary representation using the method of Sela et. al. [Sela 2004]. Incisions were made on three different internal organs.

2.3. Carving Objects in Spatial Decomposition Representation

Spatial decomposition represents solids volumetrically using similarly shaped discrete primitives. These primitives are often tetrahedral meshes or voxels. Carving

volumes represented in this manner is simple to perform. First the region that bounds the path of the carving tool as it performs a cut is determined. All volumetric primitives that fall completely within this volume are removed. Primitives that straddle the border can be removed, left untouched, or subdivided depending on the method.

2.3.1. Voxels

Voxels represent a solid as a three dimensional array of regularly sized cuboids. They are analogous to pixels in an image. Generally a set of voxels bounds the solid of interest, and each voxel has an associated scalar value defining the density of the solid at that voxel, or more simply a Boolean value indicating whether the voxel is inside the shape or outside.

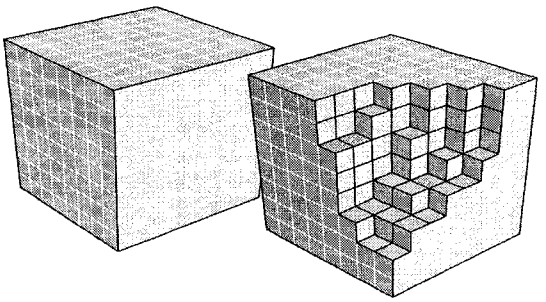


Figure 7: Diagram of a voxel set. Left is an entire set of voxels. Right is the set with some voxels removed [Engel 2004].

Voxels are often used in representing segmented medical tomography images, such as those generated through Magnetic Resonance Imaging (MRI), or Computerized Tomography (CT) scans. These techniques generate images that represent slices of the patient or object being scanned with each pixel intensity representing different levels of attenuation as an x-ray signal passes through the tissue (in the case of CT scans) or Radio-Frequency signal amplitude (in the case of MRI scans) . The images are generated in parallel slices of a constant thickness. By stacking these slices in order one can

generate a volumetric voxel representation of the object or patient that directly matches the data [Pham 2000][Schmidt 1999].

When a 3D volume is represented using voxels, the direct rendering of the voxels has a low fidelity to the original shape the voxels represent. The shape is blocky, unsmooth, and has inconsistent shading, and these problems are exacerbated by zooming up close to the object surface. For these reasons the volume represented by voxels are not often rendered directly but instead using a different technique. For example the volume could be rendered a boundary representation using various surface extraction methods such as Marching Cubes, or rendered using surface-splatting techniques [Zwicker 2001].

Carving volumes represented as voxels has been researched extensively, especially in the medical simulation field. Often the research combines visual rendering (methods for displaying the volume being manipulated) and haptic rendering (being the method of generating the correct forces in the use of hand-held input devices). Voxels provide a simple and fast method for computing simple set operations.

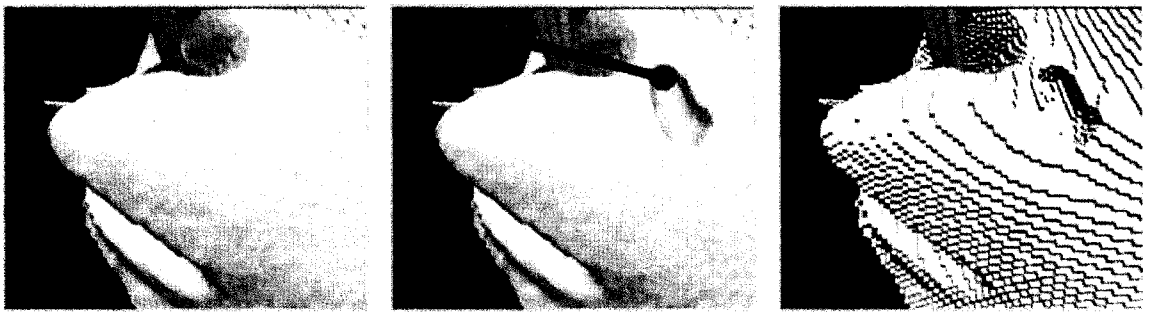


Figure 8: Carving method of Morris et. al. Voxels are used to represent volume, while a mesh is used for display [Morris 2005].

In [Morris 2005] and [Morris 2004] the authors describe a framework for the simulation of bone surgery which involves both visual and haptic components. In order to represent the bone volume they store the bone data as voxels, but generate a triangle

mesh to graphically display the bone. When voxels are removed by carving the locally affected area is re-meshed. Their triangulation algorithm is brute force: create all possible triangles between neighboring voxels and remove the triangles that are found to be duplicated or are below the object surface. Our method is similar in that we also use voxels as a base representation and maintain an associated triangle mesh for display, however our local re-triangulation method follows a well-defined process rather than creating all possible triangles and using heuristics to remove the unneeded triangles. Having a well-defined process allows more rigorous analysis of the behaviour and a more predictable efficiency.

In [Agus 2003] the authors present a system for the haptic and visual simulation of temporal bone surgery which is based on a voxel representation. In addition to bone matter, bone dust, debris, and water are represented using a particle system during the simulation. Rendering of the voxels are done using a direct volume rendering technique, in which the voxels are rendered directly using the GPU of the graphics card. The voxels use a "proxy-geometry" of object-aligned slices, treated as a geometry primitive in the graphics card, which are accumulated back-to-front. Shading and opacity for individual voxels is computed by shader and vertex programs on the graphics card itself. The method is fast since most of the visualization computation is offloaded to the GPU. One drawback of this visualization method is that it still suffers from terracing artifacts. In addition since three separate axis-aligned stacks of volume slices must be used so that the stack most perpendicular to the viewing direction can be chosen, when a new stack is selected for viewing when the viewpoint changes abrupt and unsightly changes to the texture can result [Hadwiger 2002].

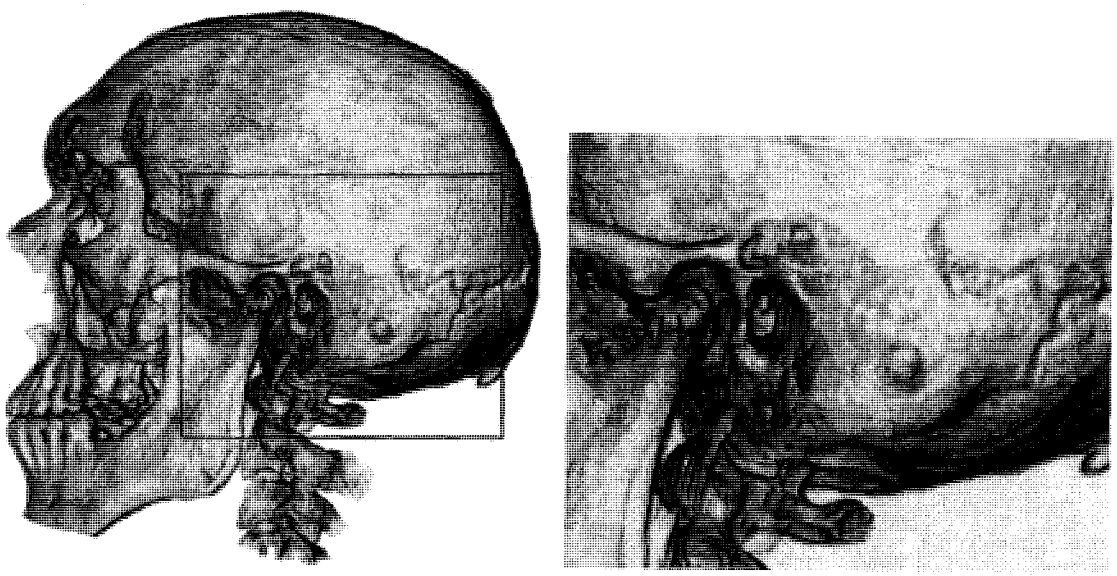


Figure 9: Representing a 3D object using axis aligned slices as done by Agus et. al [Agus 2003].

In [Acosta 2007] the authors again represent the bone being cut as voxels, with each voxel storing a density value for the amount of bone in the voxel. The tools they simulate for carving are surgical bone burrs and drills. In order to compute the removal of volume from the bone the authors use a regular set of points to represent the cutting head of the tools, and supports cylindrical and spherical heads. This set of points is generated by voxelizing the head, assigning an erosion factor to each voxel, and taking the centres of the voxels as the set of points. Only the cutting head can interact with the surface being carved. When cutting, each point on the cutting head is tested for whether it intersects a voxel having density greater than zero. For all such voxels their density is reduced based on the erosion factor of the point.

Haptic rendering is performed using a similar set of points, but which are only distributed over the surface of the cutting head. Visual rendering is performed using a direct volume rendering texturing approach similar to Agus et al. but performed using a

single 3D texture rather than rendering in object-aligned slices. Again shading calculations are performed on the GPU through fragment programs. The opacity of elements in the 3D texture is taken from the density values of the voxels.

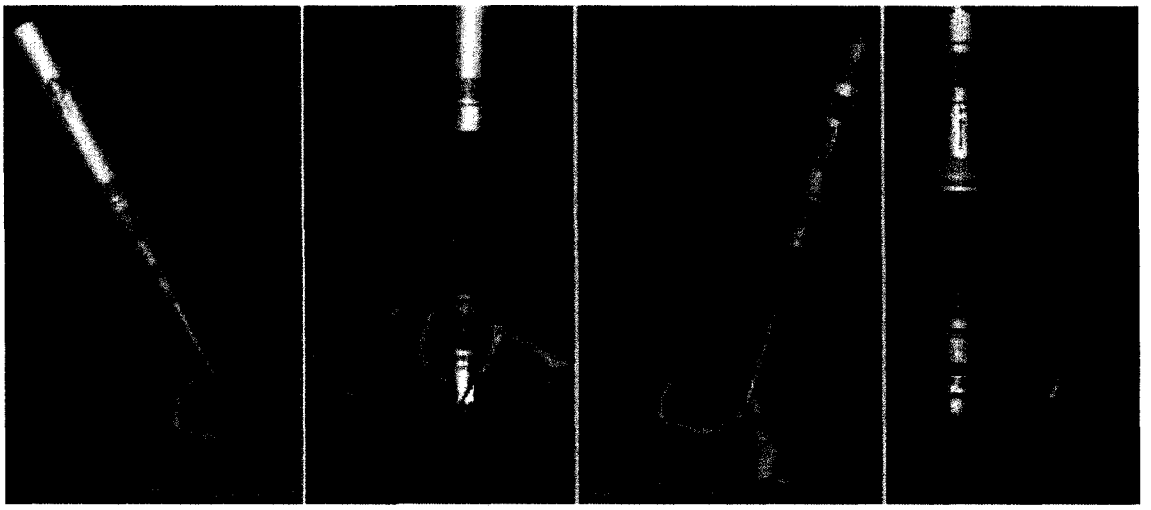


Figure 10: Representing an object using voxels as per Acosta's method [Acosta 2007].

2.3.2. Tetrahedral Meshes

Tetrahedral meshes represent an object by dividing its volume into a number of tetrahedrons. Neighboring tetrahedrons share vertices and edges. Tetrahedral meshes have an advantage over voxels as a volumetric representation in that they can represent volumes more accurately and smoothly, since the boundary of the tetrahedron mesh is automatically a triangle mesh. Tetrahedral meshes can be used in Finite Element Methods (FEM) that use piecewise linear elements since they are a discretization of the volume into 3D simplexes, which are a simple way to decompose the FEM problem.

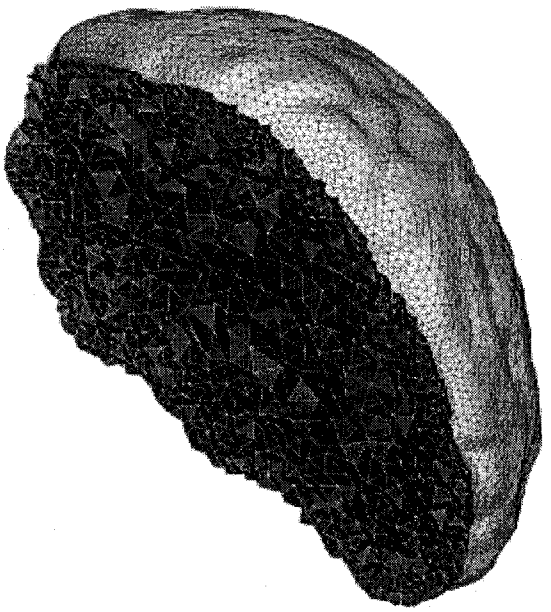


Figure 11: A 3D object represented using a tetrahedral mesh [Ito 2004].

Methods for generating a tetrahedral mesh of a volume aim to represent the surface of the volume as smoothly as possible, and represent the inside of the volume consistently. There is some research in using tetrahedral mesh representation for carving simulation. In [Picinbono 2001] the authors propose a method that models the tissue as a tetrahedron mesh and models cutting forces using Finite Element Methods (FEM).

2.4. Carving Objects in Displacement Fields Representation

Displacement fields are a composite representation of an object. The object is represented using a supporting surface S over which a vector field V is given. The surface S provides the rough base shape of the object, while the vector field V adds the details of the shape. The supporting surface can be a triangle mesh, or an analytical surface.

In practice the displacement field is rasterized into a Discrete Displacement Field (DDF). The supporting surface is sampled into a number of points, and for each point a normal vector and a scaling factor is defined. The set of points displaced by the vector field defines a new detailed surface F.

Discrete displacement fields were first introduced by Ayasse and were used to simulate Numerically Controlled (NC) milling and grinding, used in the manufacturing processes [Ayasse 2003]. NC processes use a computer controlled milling or grinding machine which consists of a table to which the item to be cut is affixed and a carving tool which the machine moves over the item to cut away pieces. The item being cut is referred to as the workpiece. The path of the tool is controlled by a program defined in the computer. The tool follows the instructions of the program to repeatedly cut away parts of the workpiece until the desired shape is achieved.

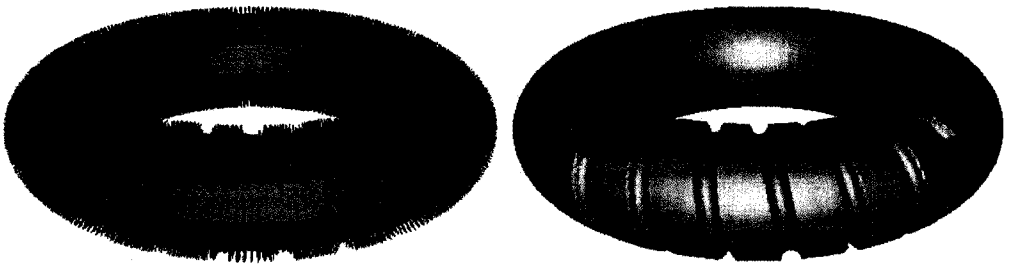


Figure 12: Simulating milling using displacement fields [Ayasse 2003].

In Ayasse's method the supporting surface is a triangle mesh. The mesh is sampled into a discrete set of points. The direction of the displacement vector for each point is interpolated from the boundary vertex normals of the triangle that the point is from. Carving is performed by calculating the intersection of the carving tool with the displacement vectors as a scale value. For each intersected vector the minimum of the intersection scale and the vector's previous scale is used as the new scale. Basically, each vector is cut shorter as if it were a stalk of straw in a field.

To generate a pleasing visualization, when the points are sampled from the support surface they are arranged according to a raster grid. A predefined set of triangles

can be used to join the points in this initial grid, and since the displacement vectors define a similar topology the same triangles can be used to represent the new detailed surface *F*.

In [Ren 2006] the authors extend Ayasse's representation to use point rendering rather than use a boundary representation for NC grinding. The authors use *surfels* to represent the new surface elements. The support surface *S* in this case is taken to be a CAD model of the correct final result of the grinding process, and the detailed surface *F* is the stock workpiece. The support surface is sampled using a Layered Depth Cube (LDC) approach to generate the discrete set of support surface points. The normals at these points are intersected with the stock workpiece to generate the displacement vectors. The model for computing the amount of material removed in a pass of the tool is based on the assumption of an elastic deformation allowing the use of the classic Hooke's law on the displacement vectors in a localized contact region of the tool, taking into account the angle of the tool in that region. Error is estimated by comparing the length of the displacement vectors after making a cut against the cutting tolerance.

The benefit to DDF methods are that very small changes in volume can be accurately represented. However the methods are not applicable to general carving since the shapes that result from this form of carving must closely follow the supporting surface shape. For example, for a planar supporting surface it is not possible to represent a cut parallel to the plane that would separate a displacement vector into two parts. As well special problems can arise in which deep carving in the same area can cause displacement vectors to intersect, and lead to triangles in *F* that have reversed orientation.

2.5. Comparison of Carving Techniques

As presented in the previous sections, there are a number of different carving techniques in the literature. The following table attempts to summarize the differences and similarities between them for comparison.

The *Update Speed* column in the table is used to describe the rate at which the technique can update the display while the user is performing model-changing carving actions to the object on consumer-grade hardware. Often qualitative terms are used in this column to describe the speed of the technique since a number of the techniques presented in the table do not provide quantitative measurements. The possible values in the column are:

- *non-animated*, meaning updating the display cannot be achieved at a rate of 24 frames per second (fps)
- *partly-animated*, meaning updating the display can be achieved at about 24 fps, but not significantly higher (thus other competing concurrent system-intensive tasks such as physics modeling may push the frame rate below 24 fps)
- *fully-animated*, meaning rates significantly higher than 24 fps are possible. In such cases the comments describe the justification for the term assigned.

The justifications for which term is assigned are largely taken from the paper describing the methods.

Method	Volumetric Representation	Rendering Technique	Update Speed (change rendered to screen)	Comments
CSG	Analytical Primitives	Ray Tracing	Non-animated	Complex objects require very large trees to represent and so more complex to render. Ray tracing is still fairly slow
Volumetric CSG [Liao 2002]	Analytical Primitives	Voxel Data Set	Fully-animated	Direct volume rendering is possible. Suffers from artifacts
Radial Basis Functions [Turk 2002]	Set of Implicit Functions	Triangle Mesh via. Marching Cubes	Non-animated	Changes to the surface require solving a the large system of equations that represents the surface
Metaballs [Turk 2002]	Set of Implicit Functions	Triangle Mesh via. Marching Cubes, Raytracing	Fully-animated	Difficulty representing flat surfaces and crisp cuts
Scalar Trivariate uniform B-spline functions [Raviv 1999]	Set of Implicit Functions	Triangle Mesh via. Marching Cubes	Partly-animated (16.72 fps for 128^3)	Complicated
Boolean operations between meshes [Thibault 1987][Naylor 1990]	Mesh	Mesh	Unknown	Very complex to implement. Rounding errors propagate throughout cutting process.
Discontinuous FFD over mesh [Sela 2004]	Mesh	Mesh	Unknown	Only supports slicing and general carving
Morris' burr simulation [Morris 2005][Morris 2004]	Voxel	Mesh	Fully-animated	Re-trianglulation is brute force, and so performs unnecessary computation
Agus' burr simulation [Agus 2003]	Voxel	Direct Volume Rendering	Fully-animated	Visual artifacts are visible, and are especially apparent from certain viewpoints
Acosta's burr simulation [Acosta 2007]	Voxel	Direct Volume Rendering	Fully-animated	Requires that the entire working volume can be placed into video card memory as a 3D texture
Discrete displacement fields [Ayasse 2003]	Mesh and Vector field	Mesh	Fully-animated	Only suitable for small changes that are generally perpendicular to the vector field
Robotic grinding simulation [Ren 2006]	Mesh and Vector field	Surfels	Fully-animated	Only suitable for small changes that are generally perpendicular to the vector field
Dynamic Ball Pivoting Algorithm	Voxel	Mesh	Partly-animated	Our method

Table 1: Comparison of Carving Techniques

Chapter 3 Volume Carving Simulation System

Our system consists of a series of different activities that must be performed to effect carving. They can be divided into two main stages:

1. an initialization stage, where a new object that has never been carved before is converted to be suitable for carving, and
2. a carving stage, where carving can be iteratively performed.

These two stages are shown as UML activity diagrams in Figures 13 and 14.

1. Initialization Stage

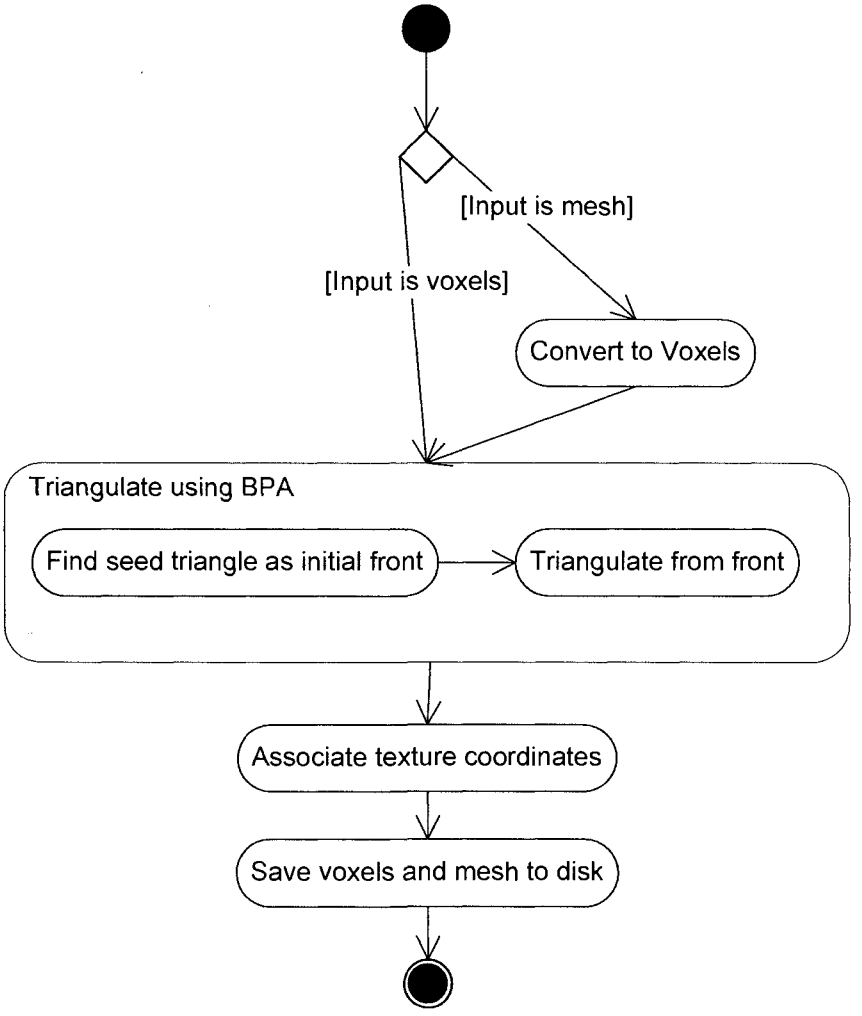


Figure 13: Initialization Stage

The initialization stage is only needed to prepare a new 3D object for carving. The output of this stage is a representation of the 3D object that can be saved to disk and used repeatedly for carving later. There are two representations of input acceptable at this stage: voxels or mesh. The input may already be a voxel representation if it is the result of medical data processing, for example MRI or CT scans. If the input is a mesh it must be converted to voxels before further processing. This conversion process is explained in detail in section 3.3.

With the voxel representation established, the next step is to create a new triangle mesh from the voxels using the BPA algorithm applied to voxels. The specifics of applying BPA to voxels are explained in section 3.7. It may seem superfluous to convert an input mesh to voxels and then back to a mesh again, but it must be done for two reasons:

- The position of a mesh vertex must correspond with the position of a voxel's centre in order to re-triangulate the mesh after a cut
- The original mesh may not satisfy an important constraint that our dynamic extension to the BPA requires of meshes: that for each face in the mesh, the ρ-ball that circumscribes the vertices of the face and whose centre is outside the face must contain no other vertices.

Once the voxels and the associated BPA mesh have been generated, texture can then be associated if texture is required for the object. Our system supports a 3D internal texture, a 2D external texture, or both. Associating the 3D texture is a process of aligning and scaling the texture to fit the object. Associating a 2D texture is more complicated,

and involves using an external tool to wrap the texture to the object. These processes are explained in Chapter 3 section 8.

Once the optional texturing task is complete, the combined voxel/mesh representation can be saved to disk so that it can be loaded later without having to perform all the initialization processing again.

2. Carving Stage

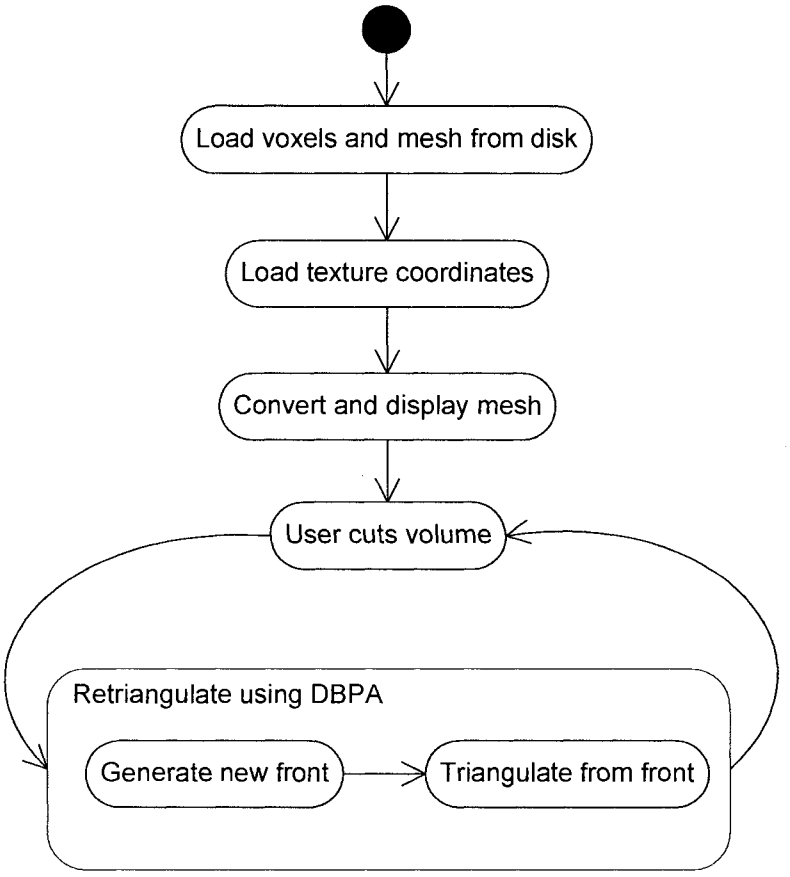


Figure 14: Carving Stage

The input to the carving stage is the output of the initialization stage: a voxel set and its associated BPA mesh, and optional texture associations. Once these are loaded from disk, the system enters a loop in which three steps are performed:

1. the mesh is converted for display

2. the user interacts with the object and performs a cut
3. the object is modified to reflect the cut

The first step is necessary to convert the system's internal mesh format into a format suitable for the underlying graphics library. The internal format is a graph-like structure which also contains extra information not needed for rendering but necessary for re-meshing purposes. The underlying graphics library OpenSceneGraph has its own mesh representation that must be conformed to when displaying.

Once converted the user can rotate and move the object and zoom in and out. When the mouse is moved over the object a carving tool is moved along the surface, and the user can click to make cuts, or enable an automatic carving mode so that each movement cuts the surface of the object.

The carving tool consists of a head and a handle. The head interacts with the object while the handle is merely a visual aid. Each time the user carves the object the voxels within the volume of the carving tool's head are removed from the voxel set of the object. Based on these removed voxels only the affected area of the mesh is re-triangulated using our dynamic extension to BPA as explained in Chapter 3 section 6 below. From a high level, the extension determines a new BPA front that bounds the affected mesh triangles and vertices, removes those triangles and vertices entirely inside the front, and re-closes the front using BPA. The idea is shown in Figure 15.

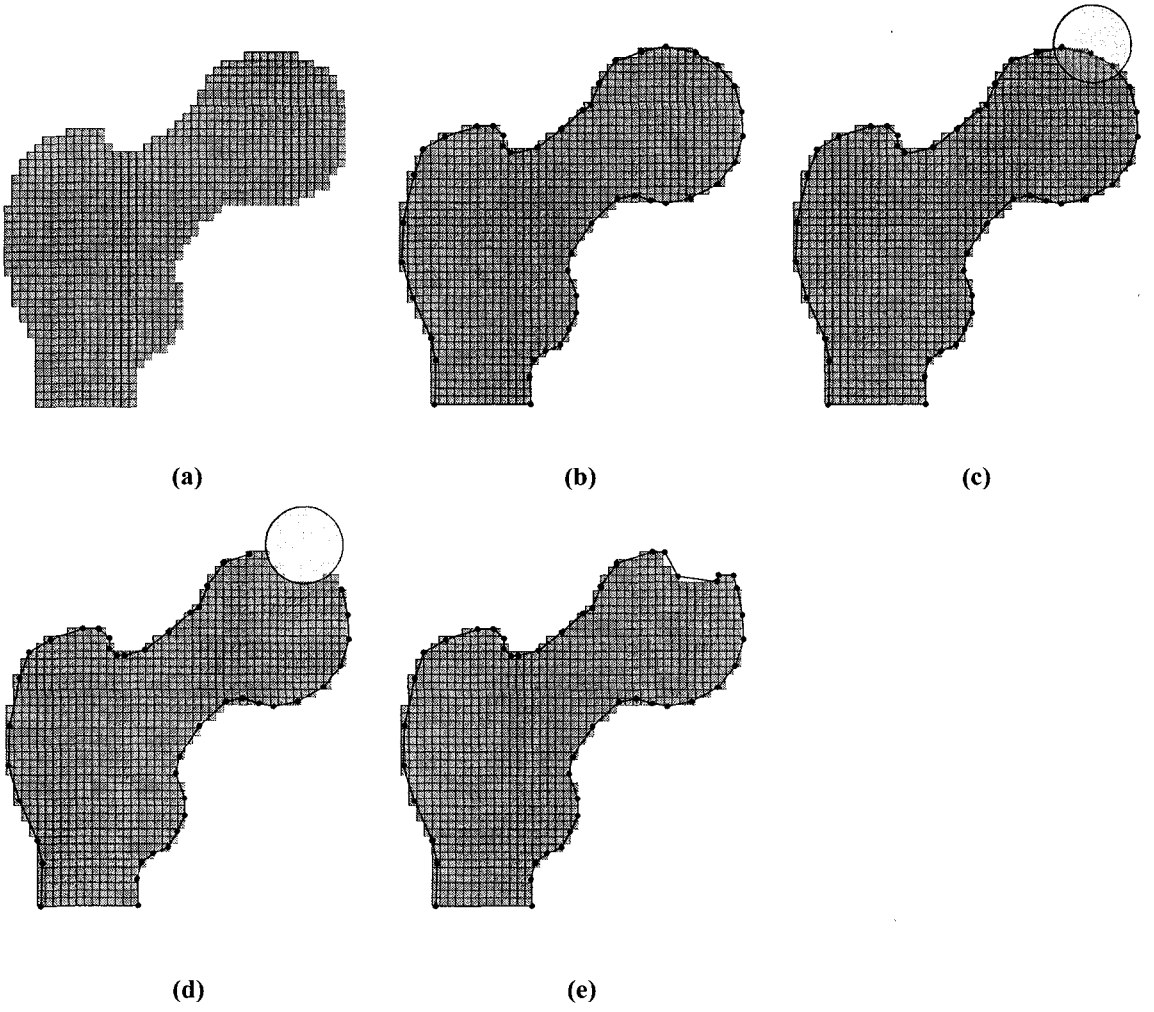


Figure 15: A 2D depiction of our solution. The object to be carved is initially represented as voxels, or converted to voxels if necessary as shown in (a). Next a corresponding triangle mesh bounding the voxels is created as shown in (b). When a cutting tool is moved so that it intersects with the object being carved as in (c), then the voxels within the intersection are removed as well as the mesh elements as shown in (d). Finally the mesh is re-closed by creating the new mesh elements on the object boundary in the locally affected region as shown in (e).

3.1. Review of the Ball Pivoting Algorithm (BPA)

Since a large portion of this work is dedicated to the Dynamic Ball-Pivoting Algorithm, which is a novel extension of the Ball Pivoting Algorithm of Bernardini et al.

[Bernardini 1999], the reader should have a thorough understanding of the BPA. This section provides an overview of the BPA algorithm.

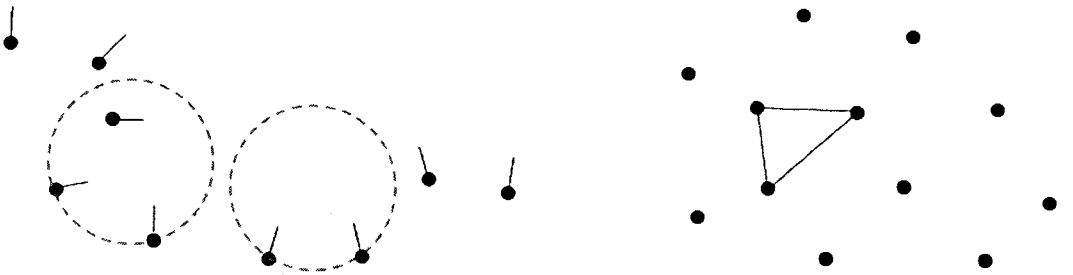
The BPA algorithm was created as a solution to the surface extraction problem: the problem of finding a surface representation of a volumetric data set. There are many algorithms that have been developed to solve this problem. They can generally be divided into three categories: spatial partitioning methods, iterative improvement methods, and region growing methods

Spatial partitioning methods generally divide the input data or points into cells. The cells are searched to find the cells that lie on the boundary of the shape, and triangles are generated based on these cells. These methods include the well known Marching Cubes method and Marching Tetrahedrons [Lorensen 1987] [Muller 1997].

Iterative improvement methods generally start with an approximation to the shape, and then iteratively improve the approximation. Examples of this category include ShrinkWrap [Overveld 2004], which begins with a sphere bounding the shape that is deformed piece by piece until it matches the input shape, and SurfaceNets [Frisken Gibson 1998], which begins by fitting a connected graph into the set of boundary voxels of the shape, after which edge energy is globally minimized so that vertex positions are relaxed while still constrained by their original voxels.

Region growing methods begin with a seed polygon that is determined to be on the surface of the shape, and then grow adjacent polygons outwards from the seed over the data set until the shape is fully covered by the mesh. Examples of this category include Marching Triangles [Akkouche 2001], which pioneered this paradigm, as well as Spiraling Edge [Crossno 1999]. The BPA falls into this category [Bernardini 1999].

In order to visualize our volume, we use the Ball-Pivoting Algorithm. The BPA is an efficient method for computing a triangle mesh from a point cloud. Like marching triangles, it is a region-growing shape reconstruction algorithm. It makes use of a ball of fixed radius ρ to determine which points are surface points, called a ρ−ball. The BPA begins with an initial seed triangle such that if a ρ-ball is touching all three points of the triangle it contains no other surface points. The edges of this triangle form the boundary of the expanding mesh called the front (F) which is the set of edges from which the triangle mesh can expand. Each edge in the front has an associated ball-centre which is the position of the centre of the ρ-ball when the ball had touched the edge's vertices. The BPA initialization process is shown in Figure 16.



(a) (b)

Figure 16: BPA initialization. In (a), the BPA initialization is shown in 2D, with vertices depicted as solid circles and surface normals depicted as lines adjacent to the vertices. A ρ-ball must be chosen that circumscribes two points such that it contains no other points and whose centre is outwards with respect to the surface normals. The position of the dashed circle on the left in (a) is invalid, while the position on the right is valid. The same process is performed in 3D but with three points. Once chosen, the adjacent vertices form the initial triangle and front of the algorithm, as shown in (b).

Each edge in the front is pivoted around by the ρ-ball. Beginning from the ball-centre that is associated with the edge, the ball is rotated such that the edge is the axis of rotation. The first or earliest point that the ball touches along it's trajectory that is not

processed, along with the pivoting edge, form a triangle that is added to the mesh. The front is updated by removing the edge we pivoted about, and adding the two new edges just created. If the ball is pivoted and no point is touched then the pivoting edge is marked as a boundary edge and is no longer considered as a candidate for pivoting around. Figure 17 demonstrates the general idea of the BPA in 2D, and Figure 17 depicts the process in more detail.

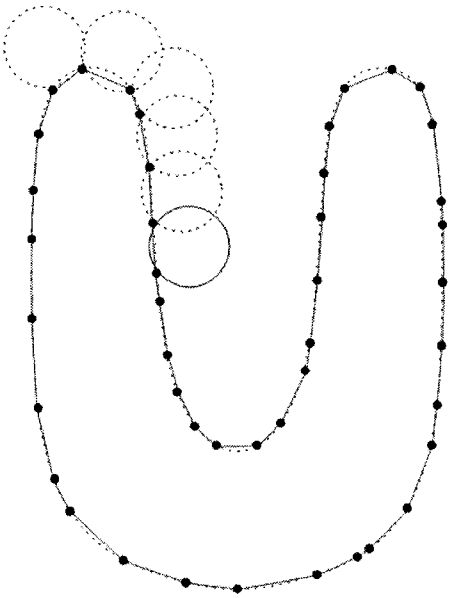
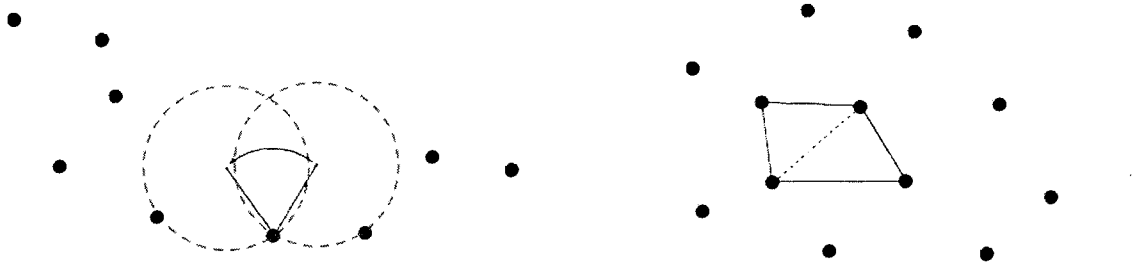


Figure 17: Ball Pivoting Algorithm in 2D. A circle of radius rho pivots from sample point to sample point, connecting them with edges.



(a) **(b)**

Figure 18: BPA after initialization. In (a), the ρ-ball is pivoted around an edge (shown as a point in the 2D diagram) until it strikes a new point. The new point and the edge rotated about form a new triangle in the mesh as shown in (b). The new front is now the perimeter around both triangles.

Each time the a new point is added to the mesh that point is marked as processed, and only points that are not-processed and that are not in the front are considered for adding to the mesh. This is to prevent the algorithm from creating a non-orientable

manifold. The front begins as a simple cycle of edges, but as edges are added the topology can become complex. The front will eventually consist of more than one cycle of edges (these cycles are referred to as loops in [Bernardini 1999]). The BPA uses two topological operators to maintain the front: join and glue. A join is performed when the pivoting ball touches a new point. It removes the pivoting edge from the front and adds the two new edges. However in creating these two new front edges we may have added coincident edges to the front; there may already have been a front edge that joined each of the new edge's points. When a coincident edge is encountered, the glue operation is applied to remove the edges. After removing the edges, however, the front must still be a set of cycles; no element of the front can be a simple path. Glue deals with four cases of coincident edges:

1. The coincident edges are a complete cycle. In this case the cycle is simply removed from the front.
2. The coincident edges are adjacent and in the same cycle. In this case the edges are removed and the cycle is shortened.
3. The coincident edges belong to different cycles. In this case the edges are removed and the cycles merged into one.
4. The coincident edges belong to different the same cycle. In this case the edges are removed and the cycle is split into two cycles.

3.2. Implementation of Discrete Voxel Data Structure

In our implementation of BPA and DBPA, we actually make use of two voxel data structures. The first is used to represent the presence or absence of matter as normal,

and the second is used as a lookup table to determine whether the voxel at a certain position has already been processed during BPA or not, as required by BPA.

3.2.1. Logical Implementation

The volume representation used in our system is voxels while our display representation is a triangle mesh. We can consider that these representations exist in two different spaces, the voxel space and the real-world space.

Voxel space is a three-dimensional space S^3 defined over a bounded set of non-negative integers $S = \{0,1,2,\ldots,r-1\}$, where r is the resolution of the voxel set and can be thought of as the length of each dimension in a three-dimensional array storing the voxel set. Each vector in voxel space specifies the location of a voxel. The voxel data structure forms a function f that maps a vector in voxel space to a binary value: $f: S \rightarrow \{0,1\}$.

To be able to carve voxels using a real-world volume and to determine the real-world position of voxels for rendering, two mappings must be defined: one to convert a voxel space vector to a real-world vector, and another to convert a real-world vector to a voxel space vector.

The voxel to real-world mapping is defined when initially creating the voxel data structure. Four real-world vectors are defined with the data structure: $\vec{o}, \vec{i}, \vec{j}$, and $\vec{k}$. The $\vec{o}$ vector is the real-world position of the origin in the voxel space. The $\vec{i}$ vector is a vector that specifies the direction that the voxel-space x-axis points in as values along the axis increase, and its length is the length in real-world space of one unit along the x-axis on voxel space. Similarly, the $\vec{j}$, and $\vec{k}$ vectors specify the real-world direction and real-world change for a unit step in the voxel-space y and z axes respectively. The $\vec{i}, \vec{j}$, and

$\vec{k}$ vectors are orthogonal and non-zero. These vectors are used define a mapping from voxel space to real-world space. If $\vec{v} = (v_x, v_y, v_z)$ is a voxel space vector and $\vec{s}$ is its real-world equivalent vector then:

$$\vec{s} = \vec{o} + \vec{i}v_x + \vec{j}v_y + \vec{k}v_z \tag{8}$$

In matrix notation, this is:

$$\vec{s} = B\vec{v} + \vec{o} \tag{9}$$

where we treat $\vec{s}$, $\vec{v}$, and $\vec{o}$ as column vectors, and

$$B = \begin{pmatrix} i_x & j_x & k_x \\ i_y & j_y & k_y \\ i_z & j_z & k_z \end{pmatrix} \tag{10}$$

The real-world to voxel mapping is derived by inverting 9:

$$\vec{v} = B^{-1}(\vec{s} - \vec{o}) \tag{11}$$

Since $\vec{i}$, $\vec{j}$, and $\vec{k}$ are non-zero and orthogonal, B is a basis and so the inverse B^{-1} always exists. However, since V is a discrete space, multiple real world values map to the same voxel:

$$\underline{v} = round\left(B^{-1}(\vec{s} - \vec{o})\right) \tag{12}$$

where *round* rounds to the nearest whole number.

The $\vec{o}, \vec{i}, \vec{j}$, and $\vec{k}$ vectors are computed when an existing object is sampled for conversion to voxels. First, the resolution r of the voxel space is chosen based on the accuracy required during carving. If the object is a mesh, then the minimum and

maximum values of each coordinate of the vertices of the mesh are found: x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, and z_{max}. The vectors are then defined as follows:

$$\begin{aligned} \vec{o} &= (x_{min}, y_{min}, z_{min}) \\ \vec{i} &= (\frac{x_{max} - x_{min}}{r-1}, 0, 0) \\ \vec{j} &= (0, \frac{y_{max} - y_{min}}{r-1}, 0) \\ \vec{k} &= (0, 0, \frac{z_{max} - z_{min}}{r-1}) \end{aligned} \tag{13}$$

By using this definition the aspect of an individual voxel is not necessarily square. Instead the aspect is compressed in the direction that the object is most compressed in. Since the resolution in each voxel space axis is constant, this means that there are less wasted empty voxels since the resolution along that axis is used to represent as much object as possible.

3.2.2. Physical Implementation

In order to use as little memory as possible and so increase the resolution of the dataset that can be used with our system, we represent each voxel using a single bit. The mapping function f: $S \rightarrow \{0,1\}$ is stored as a lookup table and is physically implemented as an array of words on a 32-bit architecture. The length of the array in words is then $l = floor(r^3) + 1$.

The bit index for a specific a voxel $\vec{v} = (v_x, v_y, v_z)$ is computed as $p = r^2 v_z + r v_y + v_x$. Setting the value for a specific voxel is done by finding the array element at $floor(p/32)$ and if the value being set is one, setting the element to be the bitwise OR of it's previous value and the bitmask generated by $\frac{2^{31}}{p \bmod 32}$, and if the value being set is

zero, setting the element to be the bitwise AND of the bitwise inverse of the bitmask. The division calculation is performed using bitwise shifts for speed.

Likewise, retrieving the value for a specific voxel consists of determining its bit position p, generating the bitmask previously mentioned, and performing a bitwise AND of the array value and the bitmask. If the result is greater than zero, the bit is set, otherwise the bit is unset.

3.2.3. Finding voxels within distance of a real-world point

One of the most time critical operations when the user is carving is to determine all of the voxels that are within range of a pivoting ball. At most this region is all the voxels within a 2ρ radius of the midpoint of the pivoting edge. This operation is performed for each triangle generated when polygonizing a voxel volume.

To compute this set of voxels a voxel space bounding box enclosing the region is computed, and then all the voxels in the bounding box are checked whether they are within range. Since the voxel space is a discrete space, the voxel-space bounding box is slightly larger than the real-world space in order to enclose all the necessary voxels.

3.3. *Voxelization of Input Meshes*

Often 3D objects are represented as meshes. In order to support the carving of input meshes, we must convert the mesh to a voxel data set before carving. We use the following method for the conversion. It assumes that the mesh is closed and connected.

First, the voxel dataset is initialized so that the $\vec{i}$, $\vec{j}$, and $\vec{k}$ vectors are aligned with the real-world x, y, and z axes respectively, the size of the voxel set is large enough to contain the entire mesh, and all voxels are set to zero. For each triangle in the mesh an

axis-aligned bounding box is computed. The coordinates of the corners of the bounding box are converted from real space to voxel space. Since the bounding box is axis aligned and since the voxel space basis vectors are also axis aligned with the real world axes, it is possible to simply loop over the voxels in this region in voxel space. For each voxel in the bounding box the fast 3D triangle-box overlap test from [Akenine-Möller 2005] is applied to the voxel and the bounded triangle. All overlapping voxels are set to one. Figure 19 demonstrates this process.

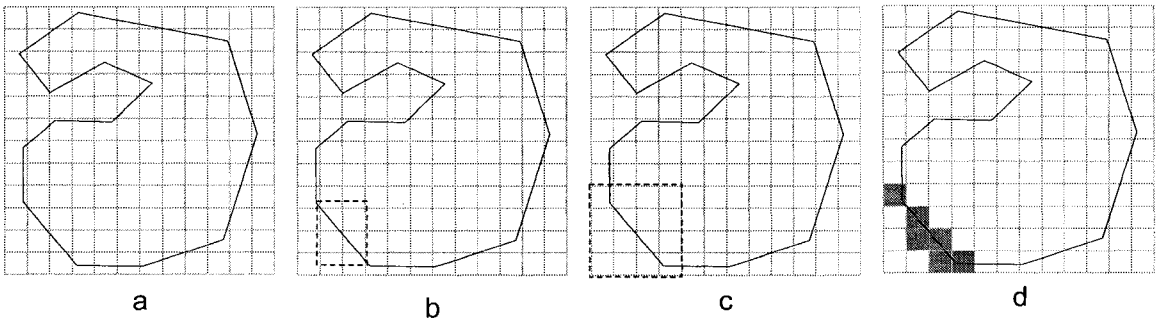


Figure 19: Creating the voxel boundary of the mesh in 2D. a shows the initial mesh and voxel set. b shows the bounding box for an edge, which would bound a triangle in 3D. c shows the bounding box in voxel space. d shows the result of filling all overlapping voxels.

Once all of the triangles are processed the voxels having value 1 form a boundary around the volume. This boundary is filled using a 3D scanline filling algorithm. The filling algorithm proceeds by first finding a seed voxel that is inside the boundary of one voxels. The seed voxel is added to a queue of seeds. Then while there are seeds in the queue, one is removed, and the column of the seed is filled until a one is hit, changing zero values to one. When a one is hit, no voxels are filled until another one is hit and filling begins again. Then new seeds are added to the queue for the previous row and next row of the seed, and the previous and next layers of the seed.

The time needed to perform the voxelization increases as the resolution of the voxel set increases, however if the resolution of the voxel set is held constant and the number of triangles in the mesh increases the performance improves since there are less box-triangle overlap tests that need to be performed, provided the triangles are not all aligned with the voxel axes which is generally the case for complex closed shapes. The performance of the voxelization is not overly critical, though, since it is only performed once per input object and voxel set pair.

3.4. Carving Tool Representation

The carving of the volume is done with a tool that consists of a handle and a cutting head. Whenever the head of the carving tool is moved over the volume, all the voxels within the boundary of the carving head of the tool are removed.

Currently the only implemented carving head is sphere shaped. When the head moves to a new position, the real-world coordinates of the head's centre is converted to voxel space. All voxels whose centre is less that the cutting head's radius distance away are set to zero, and the volume re-meshed.

3.5. Implementation of BPA

In the classic paper [Bernardini 1999] that introduced BPA there are a few details left unexplained that are necessary to implement the algorithm. This section explains our implementation of those details.

3.5.1. Testing Candidate Seed Triangles

The classic BPA must begin with an initial seed triangle whose edges comprise the initial front. The seed triangle must obey the constraints of the BPA: that a ρ-ball that circumscribes the three points and whose centre is outside the volume of the object being reconstructed does not contain any other points. To find the seed triangle different combinations of three points are tested to find a triplet that satisfy the criteria. To perform this test it is necessary to find the centre of the ρ-ball given three arbitrary points. We solved this problem as follows.

Let $\vec{a}$, $\vec{b}$, and $\vec{c}$ be the given points, and $\vec{d}$ the centre of the ρ-ball. There are two solutions for $\vec{d}$. The three points describe a unique circle in a plane through the points, and $\vec{d}$ lies on either side of the centre of this circle, in the direction perpendicular to the plane. Let $\vec{l} = \vec{p}_1 + t\vec{n}$ be a line passing through the centre of the circle and perpendicular to the plane. The direction vector n is normal to the plane and can be computed as $(\vec{b}-\vec{a})\times(\vec{c}-\vec{a})$, and normalized to unit length. The point $\vec{p}_1$ is the intersection of the bisector lines of edges *ab* and *bc*:

$$\begin{aligned} \vec{b}_{ab} &= \vec{m}_{ab} + r[(\vec{b}-\vec{a})\times\vec{n}] \\ \vec{b}_{bc} &= \vec{m}_{bc} + s[(\vec{c}-\vec{a})\times\vec{n}] \end{aligned} \tag{14}$$

where $\vec{b}_{ab}$ and $\vec{b}_{bc}$ are the bisector lines of *ab* and *bc*, $\vec{m}_{ab}$ and $\vec{m}_{bc}$ are the midpoints of *ab* and *bc*, and *r* and *s* are parameters. Now all that remains is to calculate the parameter *t* needed to substitute into line $\vec{l}$ to compute $\vec{d}$. The points $\vec{p}_1$, $\vec{a}$, and $\vec{d}$ form a right-

angle triangle with the length of *ad* being ρ. Thus $|dp_1| = \sqrt{\rho^2 - |Ap_1|^2}$. Since $\vec{n}$ is unit length, we can use $t = |dp_1|$ to compute $\vec{d}$.

3.5.2. Finding Centre of Pivoting Ball

The next important problem to be solved when implementing BPA is to compute the position of the centre of the ρ-ball when it strikes a point $\vec{p}$ during rotation about an edge. As demonstrated in [Bernardini 1999], this can be done by calculating the intersection of a ρ-ball centred at $\vec{p}$ and the trajectory of the centre of the ρ-ball. The details of how the intersection is performed are left out of the paper, however, so our solution was derived as follows.

First, the trajectory of the centre of the pivoting ball is a circle in a plane. The intersection of this trajectory and the ρ-ball centred at $\vec{p}$ can be reduced to finding the intersection of two circles: the trajectory circle, and the circle formed by the intersection of the ρ-ball centred at $\vec{p}$ with the trajectories plane, which we label *C*. The centre $\vec{c}$ of *C* is the closest point on the trajectory plane to $\vec{p}$, and the radius of *C* is $\sqrt{\rho^2 - |\vec{p} - \vec{c}|^2}$.

To intersect the two circles, the following solution is used. Let the trajectory circle have centre $\vec{c}_1$ and radius $\vec{r}_1$, and the circle of the ρ-ball have centre $\vec{c}_2$ and radius $\vec{r}_2$. The relationship between the two circles can be described by figure 20. In the figure $\vec{g}$ is one of the two points where the circles intersect, and $\vec{p}$ is the point on the line segment $\vec{c}_1\,\vec{c}_2$ such that the line segment $\vec{p}\vec{g}$ is perpendicular to $\vec{c}_1\,\vec{c}_2$.

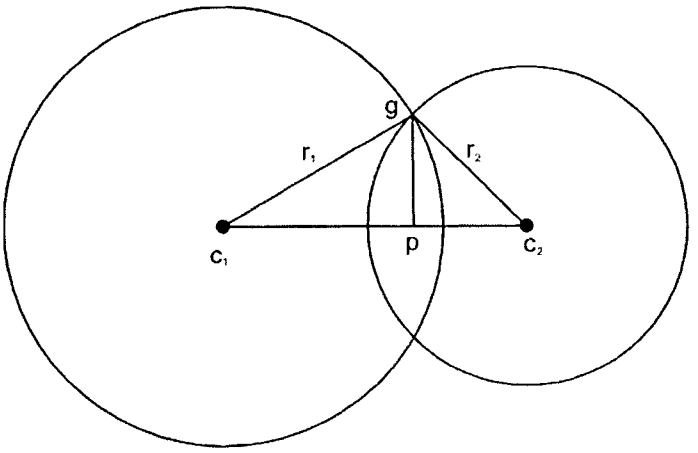


Figure 20: Geometry of Two Circles in 3D

Let l be the distance between $\vec{c}_1$ and $\vec{c}_2$, a be the distance between $\vec{p}$ and $\vec{c}_2$, and b be the distance between $\vec{p}$ and $\vec{c}_1$, and i be the distance between $\vec{p}$ and $\vec{g}$. Then

$$\begin{aligned} l &= a + b \\ r_1^2 &= b^2 + i^2 \\ r_2^2 &= a^2 + i^2 \end{aligned} \tag{15}$$

Deriving from these equations we get:

$$b = \frac{r_1^2 - r_2^2 + l^2}{2l} \tag{16}$$

which implies that

$$\vec{p} = \vec{c}_1 + \frac{\vec{c}_2 - \vec{c}_1}{|\vec{c}_2 - \vec{c}_1|} b \tag{17}$$

and

$$\vec{g} = \vec{p} \pm \left(\frac{(\vec{c}_2 - \vec{c}_1) \times \vec{n}}{|(\vec{c}_2 - \vec{c}_1) \times \vec{n}|} \right) \sqrt{r_2^2 - b^2} \tag{18}$$

Though there are two intersections, only one is the valid position of the centre of the ρ-ball when it struck the point. The correct intersection to take is determined by

computing the cross product between the vector from the centre of rotation to the projection of the point under test onto the trajectory plane, and the vector from the centre of rotation to the position of the centre of ball when it struck the point. If this cross product points in the same half-space direction as the vector representing the axis of rotation (as determined by the cosine of the angle between them) then that intersection is taken as the centre of the ball. See figure 21.

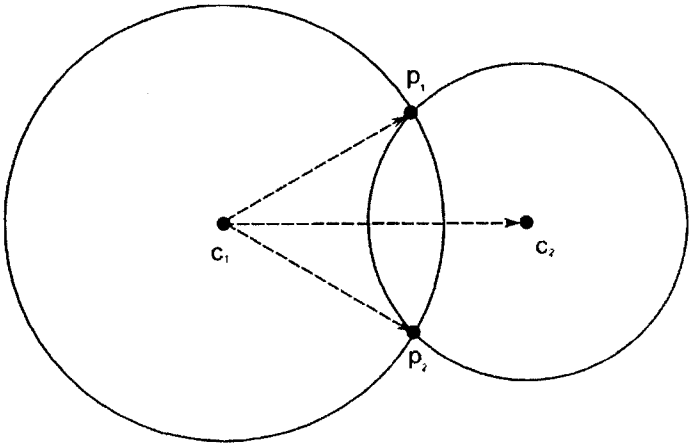


Figure 21: Determining the correct intersection to choose as the ball-centre. The point c_1 is the center of the trajectory of the ρ-ball centre, and c_2 is the point being tested. The cross product of c_1p_1 and c_1c_2 is compared with the axis of rotation vector pointing towards the viewer from c_1.

Once the correct ball-centre is found, its angle about the rotation axis with respect to the initial ball centre position is used to compare whether it is the smallest rotation or not compared to the previously smallest rotation and thus determine the first struck ball.

3.5.3. Special Case in BPA

In the original BPA algorithm there is no action taken to handle the special case in which a ball pivoting around a single edge touches not one earliest point, but multiple. This case is unlikely to occur in point sets that are generated by scanning, for which BPA was originally designed, but since voxel datasets have a very regular structure it is likely to occur in our application.

If S is the set of simultaneous earliest points, in the basic algorithm the points in S are processed in an arbitrary order. This can lead to the creation of a front that overlaps itself and that might not close properly. The problem is shown for an example case in Figure 22.

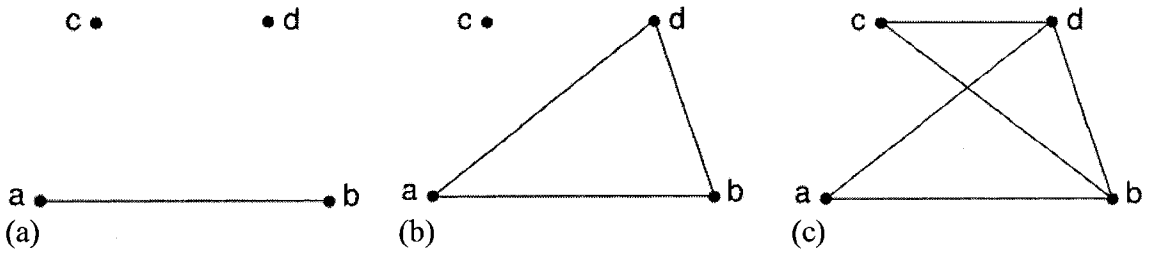


Figure 22: Special case in BPA when a pivoting ball touches multiple earliest points. (a) The edge *ab* is being pivoted around, and the ball struck the two points *c* and *d* earliest in it's trajectory. (b) Vertex *d* is processed, and the triangle *abd* is created. (c) Since the front edges are processed in an arbitrary order, we pivot around *bd* next and find that the point *c* is struck first with zero rotation. The triangle *bcd* is created, which overlaps an existing triangle.

We have implemented a simple solution to this problem. When multiple points are touched during a single pivot they are processed in order of ascending angle with respect to the pivoting edge, following these steps:

1) Let $e_{i,j}$ be the pivoting edge

2) For each s in S

    a) a = *second_vertex*(e) - *first_vertex*(e)

    b) b = s - *first_vertex*(e)

    c) compute angle between a and b

3) Sort S ascending by the computed angles

4) Set $e_{last} \leftarrow e_{i,j}$

5) while $S \neq \emptyset$

    a) $s \leftarrow$ *remove_first*(S)

    b) $f \leftarrow$ *first_vertex*(e_{last})

    c) $l \leftarrow$ *second_vertex*(e_{last})

    d) *join*(e_{last}, s, F)

e) if ($e_{s,f}$ in F) $glue(e_{s,f}, e_{f,s}, F)$

f) if ($e_{l,s}$ in F) $glue(e_{l,s}, e_{s,l}, F)$

g) $e_{last} \leftarrow e_{f,s}$

The first triangle created is between the edge and the point with the lowest angle from the pivoting edge. Each later triangle is created from the last triangle's edge which is adjacent to both the last processed point in S and to the first vertex of e. This method will not create overlapping triangles.

3.6. Dynamic Ball-Pivoting Algorithm (DBPA)

As described previously, the ball-pivoting algorithm of Bernardini et. al. [Bernardini 1999] takes as input a set of points and normals, and as output produces a triangle mesh that represents the boundary of the object. The mesh so generated has no explicit relationship with the point set that generated it, which we call the *underlying point set.* Since this is the case, if one wishes to see the effect on the boundary mesh of removing points from the underlying point set the BPA algorithm must be re-run on the entire point set to see the effect.

We have extended the BPA so that the removal of points in the underlying point set can be reflected in the mesh in an efficient manner by only applying BPA to those areas of the mesh that are affected by the removal of points. This allows us to simulate carving efficiently. We refer to our extension of the BPA as the *dynamic ball-pivoting algorithm* since our representation of the mesh can dynamically change based on the point set, in contrast with BPA which produces a static representation of a single point set configuration.

The basic idea of the algorithm is to begin by triangulating the initial set of points using the BPA. When points are removed from the underlying point set, for each boundary point the mesh elements surrounding the point are removed, and a new front is formed from the mesh edges surrounding the holes created in the mesh. The mesh holes are then re-closed using BPA with the new front.

Conceptually, DBPA takes as input either a set of points and normals, as with BPA, or specifically modified output of a previous iteration of DBPA. In the first case, the input is a point set and normals, and the output is a modified point set and triangle mesh, such that each boundary point in the modified point set has a reference to the mesh vertex that was generated from it, and each mesh triangle contains information as to how it was created from the front. In the second case, the input to DBPA is the output of a previous iteration of DBPA, except also including a second set of removed points. The output of this case is the input point set with the removed points removed, and a new mesh that bounds the modified point set.

Internally, the DBPA makes use of three interlinked data structures:

1. The underlying point set;
2. The triangle mesh; and
3. The front

The underlying point set represents all the points that will be used to represent the volume of the object. As with BPA, it must be possible to derive a surface normal vector for each boundary point in the point set.

When DBPA is run on the first case of input, a point set and normals, it basically follows the steps of the BPA to create the mesh. However, extra information is stored

with the data structures. With the BPA each time a new triangle is created in the mesh the vertices of the triangle correspond to a point in the point set. In DBPA a reference to the mesh vertex that was created by pivoting to touch a point in the point set is stored with that point. This reference is used later when points are removed from the point set to determine which mesh vertices, edges, and triangles are affected by the removal.

Each time a triangle other than the seed triangle is created in the DBPA, it is the result of pivoting from an already existing edge in the BPA front. The ρ-ball is pivoted around the edge until it strikes a point in the point set. In the DBPA when a new mesh triangle is created the centre of the ρ-ball when it stuck the point is stored with the mesh triangle. This information is used later when re-triangulating after points are removed from the point set.

In the second case of input, which is the result of a previous iteration of the DBPA and a set of removed points, the DBPA proceeds in two steps: the first step is to create a new front that bounds the mesh elements affected by the point removal, and the second is to follow the BPA algorithm steps to triangulate and re-close the surface using this front. The standard BPA algorithm begins with a front that bounds a single triangle and grows outwards from that front to cover the entire surface. Likewise in the second step of DBPA we will begin BPA with a more complex front and grow triangles to close the front, ultimately replacing deleted triangles with new triangles in the existing mesh. Figure 23 shows the process of creating the new front for a simple mesh when two surface voxels are to be removed.

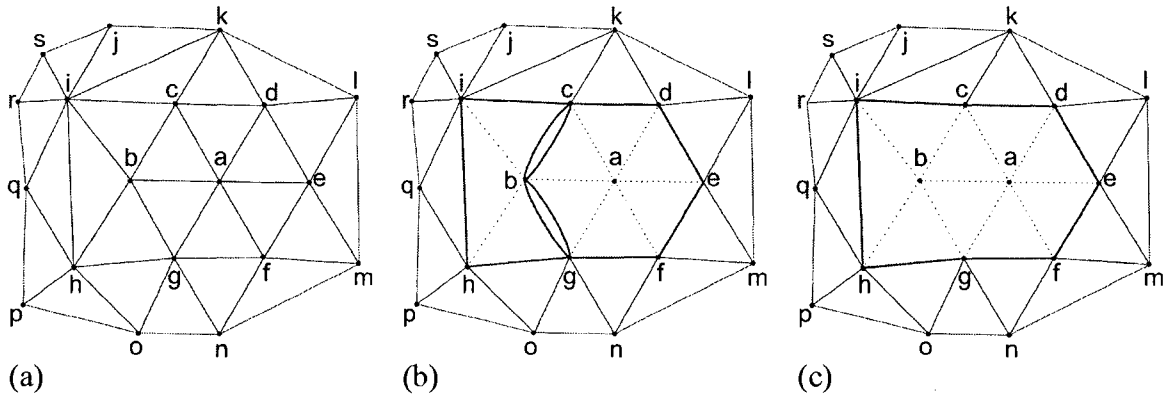


Figure 23: Steps taken to create a new front when the vertices *a* and *b* are to be removed from a sample mesh. (a) a section of a triangle mesh is shown. (b) the vertices *a* and *b* have been removed in that order. The adjacent triangles (*abc*, *acd*, *bci*, *bih*, etc.), the adjacent edges (*ab*, *ac*, *bi*, *bh*, etc.) have been removed from the mesh. Removed edges are shown as dotted lines. Two new loops have been created in the front: *bcdefgb* and *bcihgb* (shown in bold). By adding the new cycle, the front contains a coincident pair of edges for *bc* between the two cycles, and a coincident pair for *bg*. (c) the coincident edges are removed by applying glue to each coincident pair in turn, resulting in a single cycle.

In the first step, for each point p removed, if p was not a surface point then no additional steps need to be taken. If p was a surface point it has a corresponding mesh vertex and adjacent triangles. The point p is removed from the voxel set.

Since the point is removed so must the adjacent mesh triangles and the edges that depend on those underlying removed points. When these triangles are removed, a new loop in the front is created that bounds the removed triangles. We call the set of remaining edges that bound the mesh triangles removed by removing a point the *rim* of the removed voxel. The rim must be ordered so that each edge in the rim is adjacent to the last.

When creating the rim, however, we must be careful that we only remove edges that are not already part of another loop that was added to the front by removing a voxel previously in the step. For example in Figure 23 the point corresponding to the mesh vertex *a* was removed first. When *b* was later removed the correct rim includes *bc* and *bg*,

not *ac* and *ag* like expected since those edges have already been deleted and would cause overlapping loops in the front which violates the pre-conditions of BPA.

Instead we create coincident edges in the front. These coincident edges have no valid ball-centre, since they have no adjacent triangles. In the standard BPA these duplicate coincident edges may occur after a join operation, and they are removed using the BPA glue operation, which merges adjacent loops together as appropriate. We use the same glue operation to remove our coincident edges and merge separate loops into one.

The BPA front is a collection of loops of edges. With each front edge an associated ball-centre and opposite point must be stored. The ball-centre is stored so that when pivoting around the front edge the ball begins in the correct position. When each rim is added to the front, each front edge of the rim is assigned the ball-centre that was stored with the mesh triangle adjacent to the edge that was not removed, and the opposite point of that triangle. This information was stored with the mesh triangles when they were created by DBPA. This allows the ρ-ball to pivot from the correct initial position.

The set of steps taken when processing a point removed from the mesh can be summarized as follows. Let P be the set of points removed from the object. For each point p in P:

1. If p has no corresponding mesh vertex m, perform no processing on p and proceed to the next p
2. Remove the mesh vertex m that corresponds to v, and each edge and triangle adjacent to m
3. Compute the set of edges B that bounds the removed triangles, called the rim. This may be multiple cycles of edges.

4. For each cycle L_B in B
    a. Create a new empty cycle L in the front F
    b. For each edge b in L_B, create a new front edge f, and set the ball-centre of f to the ball-centre stored with the remaining triangle adjacent to b. If there are no remaining triangles, use a dummy ball-centre. Add f to L.
5. Adding edges to L in the previous step may have created coincident edges in the front. For each pair of coincident edges, use the BPA glue operator to remove the coincident edges

The procedure above depends on a method for creating the rim correctly. The rim must be a continuous loop with each edge adjacent to the previous, and must bound only the hole created by removing the triangles and edges in step 2. Our method for computing the rim of a removed mesh vertex m is presented it pseudocode as follows.

It takes as input two disjoint sets of edges: $E_{opposite}$, which is the set of edges belonging to triangles adjacent to m, but only those edges that are not adjacent to m; and E_{spokes}, the set of edges that are adjacent to m. The algorithm builds into R a set of cycles of mesh edges that bound the set of triangles that will be removed by deleting m:

```
R ← ∅
while E_opposite <> ∅
    L ← ∅
    Remove an edge e from E_opposite
    v ← first_vertex(e)
    done ← false
```

```
        while not done
                find an edge e_a in E_opposite adjacent to v
                if found then
                        remove e_a from E_opposite
                        add e_a to front of L
                        Set v to vertex v_e of e_a, where v_e ≠ v
                otherwise
                        done ← true
        end while
        if |L| < 3 or not adjacent(first(L), last(L)) then
                find edge e_s in E_spokes adjacent to v
                add e_s to front of L
        v ← second_vertex(e)
        done ← false
        while not done
                find an edge e_a in E_opposite adjacent to v
                if found then
                        remove e_a from E_opposite
                        add e_a to back of L
                        Set v to vertex v_e o f e_a, where v_e ≠ v
                otherwise
                        done ← true
        end while
        if |L| < 3 or not adjacent(first(L), last(L)) then
                find edge e_s in E_spokes adjacent to v
                add e_s to back of L
        add L to R
end while
```

It proceeds by repeatedly picking one edge from $E_{opposite}$ and building outwards the path of other edges from $E_{opposite}$ adjacent to it before and after in the mesh. If this path turns out to be a cycle then we have processed the entire $E_{opposite}$ set and the rim is that single cycle. Otherwise the spoke edges adjacent to each end of the path are found and added to the correct end of the path, and this forms a single cycle in the rim. Adding these spoke edges must create a cycle since the spoke edges are both adjacent to *m*, and each is separately adjacent to one end of the path just built.

3.7. Applying DBPA to Voxels

Our system applies the BPA to a voxel dataset. The BPA is meant to operate on a set of points and not voxels we take the points to be the centres of the voxels. Since the BPA will only generate triangles that touch surface points, we calculate the set of boundary voxels before running BPA on the voxel set, and apply BPA to the boundary voxels rather than the entire voxel set. Reducing the data size in this way decreases the processing time required to create or re-create a mesh. Boundary voxels are taken to be all voxels that have less than 26 neighbors.

In the initialization stage our system must create an initial BPA mesh around the voxel set that represents the volume. The first part of creating the mesh is finding a valid seed triangle that becomes the first front. In order to create this triangle the BPA assumes the surface normal at each point is available, or in our case for every voxel. To approximate the normal at each boundary voxel we compute vectors pointing from each neighbor voxel to the voxel, and take the average of each of these vectors.

For a voxel set, the quality of the output mesh is dependent upon the ball radius ρ. We set the ball radius ρ to be the distance from the corner of a voxel to its diametrically

opposing corner; that is, the corner that is not connected to the original corner by an edge. This length is $\rho = v_{max} = \sqrt{voxellength^2 \times voxelheight^2 \times voxelwidth^2}$. This size is large enough so that in the boundary case where one could imagine the ball to be pivoting about an edge that passes through its spherical axis, it is still large enough to touch any voxel that is within one voxel-volume of the sphere centre. It is also small enough so that in the boundary case where one could imagine the ball to be pivoting about an edge of length zero (logically, a point), it could still pass through an opening of diameter v_{max}, which is approximately 2 voxels by 2 voxels.

3.7.1. Voxel and Mesh Association

When the voxels of the object are wrapped in a mesh, either during initialization or after a cut, each vertex in the mesh corresponds to a voxel in the voxel set. Each boundary voxel that supports a vertex in the mesh, then, is associated with that mesh vertex. In our system we maintain this association via a pointer.

Whenever voxels are cut from the object with a carving tool, all voxels within the boundary of the carving tool's head are set to zero (empty). When these voxels are cleared, their corresponding mesh vertices are marked for deletion. These deleted voxels can then be processed using DBPA as described above.

3.7.2. Lookup Table

In BPA, each time a ball is pivoted around an edge e all the points within a 2ρ radius of the midpoint of e are considered to see where the centre of the pivoting ball is when it strikes the point. This is done by first calculating for each point q the intersection between a sphere of radius ρ centred at q and the ball's trajectory. These intersections

mark the centre of the rotating ball when it struck each q. Out of these the point corresponding to the intersection first along the ball's trajectory is taken as the next point. The calculations needed find the centre of the ball – if computed geometrically – involve vector projection, square root, and a number of multiplication operations.

Since in our case we are applying DBPA to an input point set with a very regular structure (i.e. a 3d grid) it is possible to improve the performance by pre-caching BPA rotation calculations in a lookup table. The voxel space is discrete. Since the edge we are pivoting around is always between two voxels, there are a finite number of possible edges that are pivoted around, if we consider the edge as a vector and hold the radius ρ of the pivoting ball constant. Since the points we must consider when pivoting are also voxels, and there are a finite number of voxels within a 2ρ distance of a given edge, there are a finite number of possible intersection points for each edge. Thus, we have created a lookup table that implements the function:

$$f_p : (A, B) \rightarrow S \tag{19}$$

where A, B are vectors in the voxel space $\mathbf{Z}^3$ and S is the set of voxel coordinates struck during pivoting and the centres of the rotating ball when it would strike the voxel. This replaces the intersection calculations with a constant time table lookup.

The A and B vectors represent the endpoints of the edge being pivoted around. The lookup table is internally indexed by the components of the vector difference between the two voxel endpoints: $B - A = (d_x, d_y, d_z)$. For each possible (d_x, d_y, d_z) value all of the positions for the centre of the pivoting ball where it strikes all nearby voxels and the real-space of the voxels are stored in a linked list. This latter value is useful to save the cost of performing a voxel-space to real-space conversion calculation.

3.8. Texturing

In order to provide realism and visual appeal the objects being carved should be textured in an efficient way. Our method allows the specification of a 2D external texture and 3D internal texture for the object being carved.

When our system first loads an object the object is textured only with the 2D texture mapped to the external faces of the object being carved. When carving is performed, however, each triangle created by applying the DBPA is texture mapped using the 3D texture rather than the 2D. When the mesh being cut is first loaded the maximum axis-aligned dimensions of the mesh are stored. When a new triangle is created using DBPA the physical coordinates are divided by the maximum initial dimensions of the mesh, giving relative texture coordinates. These relative texture coordinates are then scaled by the axis-aligned size of the 3D texture to obtain the actual texture coordinates.

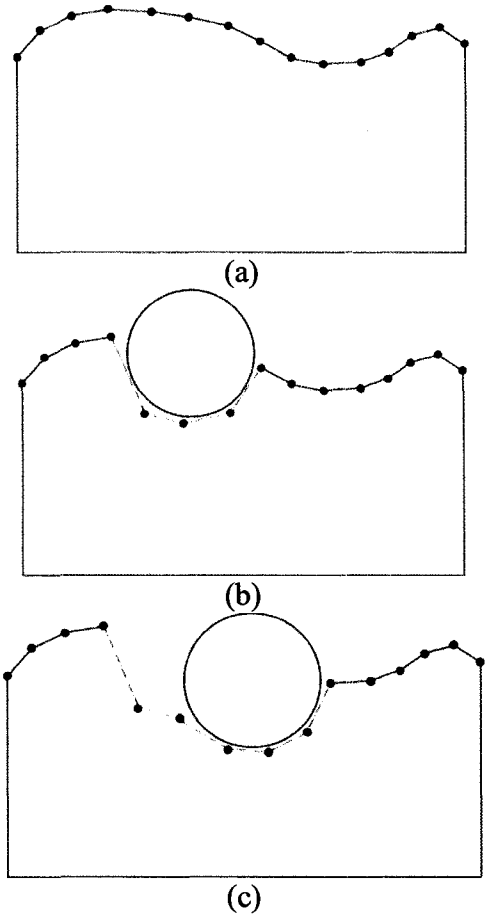


Figure 24: Combining 2D and 3D Texture Mapping. The 2D texture is solid blue while the 3D texture is dotted red

A 2D diagram explaining the process is show in Figure 24:

- In (a) the object is initially loaded and so only has a 2D texture mapped to the object. Each face of the object (shown as line segments) is flagged as not being cut.
- In (b) some of the objects vertices are removed and new faces created. The new faces are created using the 3D texture instead of the 2D texture. The voxels in the 3D texture that are chosen for the points are determined using the method explained previously and a 2D texture slice is mapped to the object from the 3D texture using interpolation between the points in the 3D texture.

- In (c) more of the objects vertices are removed including faces that were previously mapped with the 3D texture.

The methods for generating the 2D and 3D textures are explaining in more detail in the following sections.

3.8.1. 2D Texture

The problem of automatically mapping a 2D texture to the surface of a 3D shape is hard. It is difficult to automatically align the 3D texture correctly with the shape and to ensure that the texture is not pinched or stretched. In our system we avoid automatically mapping a 2D texture to the shape being carved by depending on user interaction to map the texture well. The 2D texture mapping process is shown in Figure 25. The entirety of the process shown in the figure is within the activity 'Associate Texture Coordinates' in the overall system UML diagram of Figure 25.

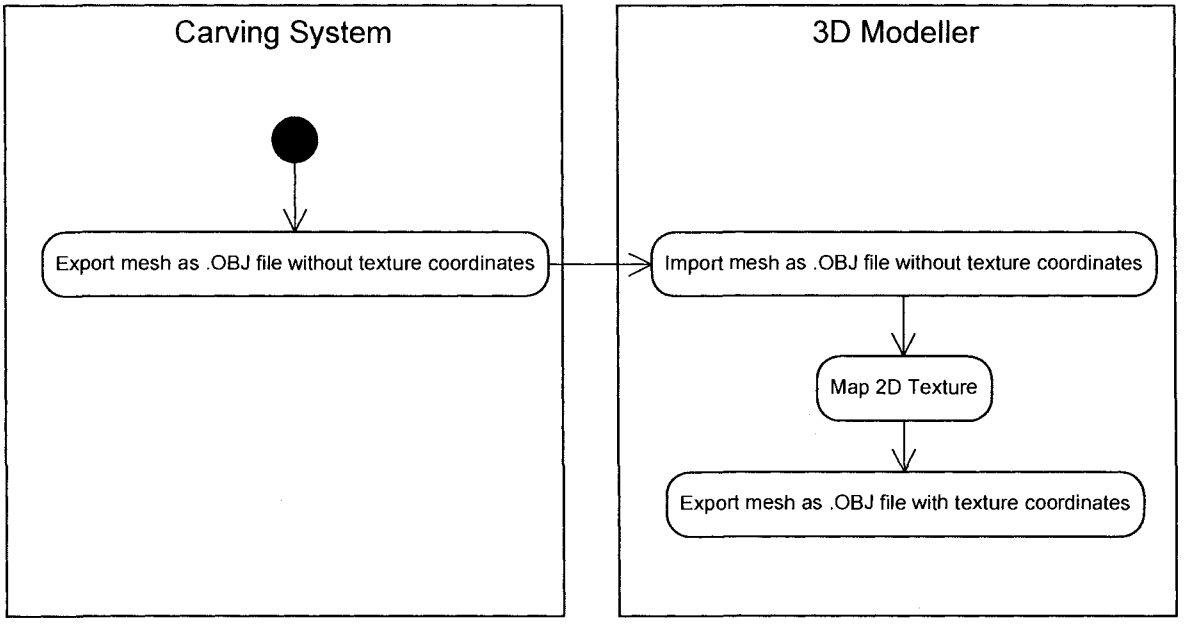


Figure 25: UML Activity diagram of the 2D texture mapping process. Swimlanes mark the responsibility of different systems.

The 2D texture mapping process begins with the BPA mesh created around the object in the initialization stage. Our system exports this mesh as a Wavefront OBJ format mesh file. This format is simple and is read by most 3D modeling tools. The user then loads the exported OBJ file into their preferred 3D modeler (such as 3D Studio Max, Blender, or Maya) and uses the tools advanced features to map the 2D texture without modifying the mesh vertices. Once the texture is mapped, the mesh is exported as a new OBJ file that is stored with the Voxel/Mesh file for the object to be carved.

When the system then loads the Voxel/Mesh file for carving, if the texture-mapped OBJ file exists then the texture coordinates from the OBJ file are read and used to map the texture of the object being carved. There is a one-to-one correspondence between the vertices in the Voxel/Mesh file and the OBJ file.

3.8.2. 3D Texture

There are a number of 3D or solid texture generation techniques. Owada et. al [Owada 2004] devised a system for making a cut in a 3D object, and a method for applying textures to the cut to simulate the internal structure of the cut object. The authors deal with three different classes of textures: isotropic, layered, and oriented textures, and discuss methods for nicely texturing cuts using these textures. There are a number of procedural methods for generating solid textures, which usually involve repeated iterations of an algorithm that introduces fractal noise or turbulence to previous iterations to generate textures. These techniques are well suited to relatively isotropic textures such as mountains or clouds [Ebert 1998]. Recently exemplar based methods have gained popularity. These methods begin with a small, often 2D texture sample and extrapolate an arbitrary sized 3D texture. Chen and Ip developed a method that begins

with a 2D texture and creates new equivalently sized 2D textures by perturbing pixels in the initial frame with Perlin turbulence [Chen 2004]. This yields a small 3D texture by stacking the new texture frames on top of each other in the Z direction. By constraining the turbulence so that as a pixel moves through the new frames it will end at the same position as it started at the far end of the stack the texture is tileable. The method of Kopf et al. begins with a 2D examplar, and uses extended 2D texture optimization techniques to synthesize 3D texture solids [Kopf 2007]. After generating the base texture a non-parametric texture optimization approach along with histogram matching is used to force the global statistics of the synthesized solid to match those of the exemplar.

For our experiments we created simple 3D textures by layering multiple copies of an image on top of itself. The 2D image was loaded and repeated as 2D slices of a 3D array. The 3D array is used as a 3D texture in the graphic library, with each image being a one "pixel" slice of the 3D texture. This is similar to the layered 3D textures used for illustration purposes by Owada et. al [Owada 2004]. This works well for the layered nature of the textures we wanted to create for our experiments. Figure 26 demonstrates our simple technique.

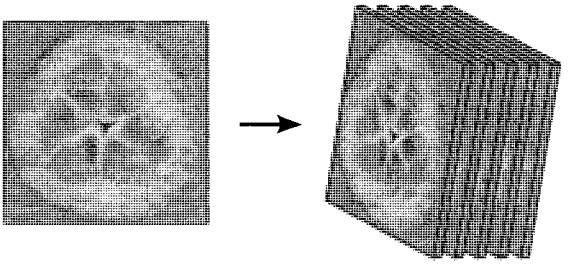


Figure 26: Creating the 3D texture. To create a 3D texture a 2D image is duplicated along the third dimension to create a 3D array of pixels.

Chapter 4 Results and Analysis

This chapter discusses and analyzes the results of applying our method for carving. In section 4.1 we provide visual results of our method for different objects. In section 4.2 we present performance results for our method and measure the speed to perform individual cuts given the number of voxels removed during the cut. Section 4.3 describes the results of experiments used to validate the accuracy of our carving method.

4.1. Visual Results

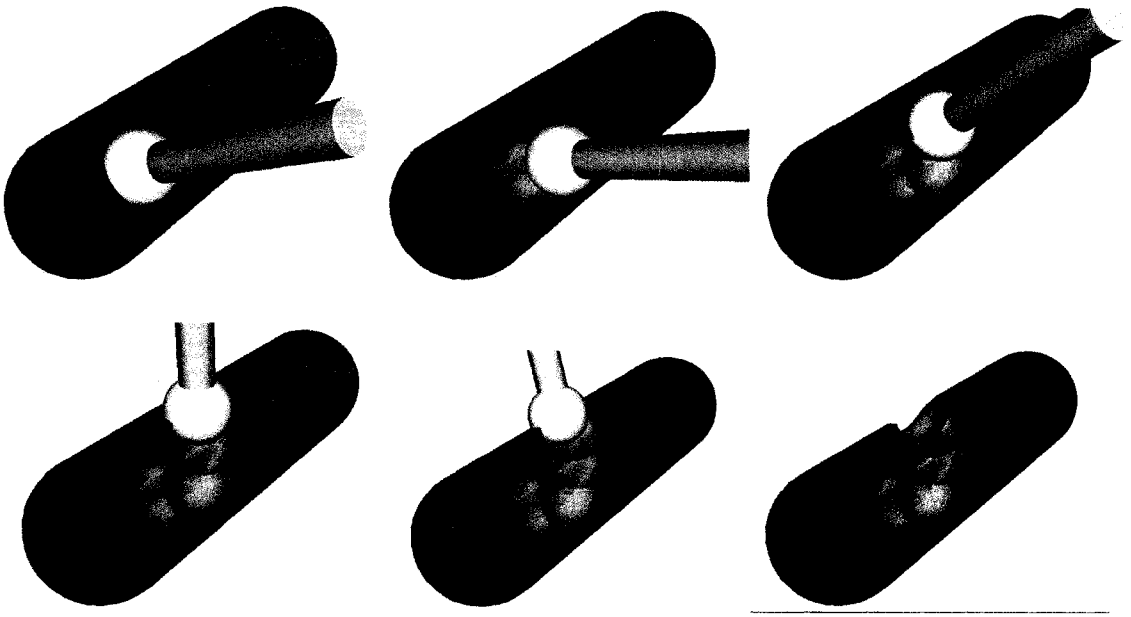


Figure 27: Demonstration of carving an externally and internally textured object. The cucumber uses an external 2D texture as well as an internal 3D texture.

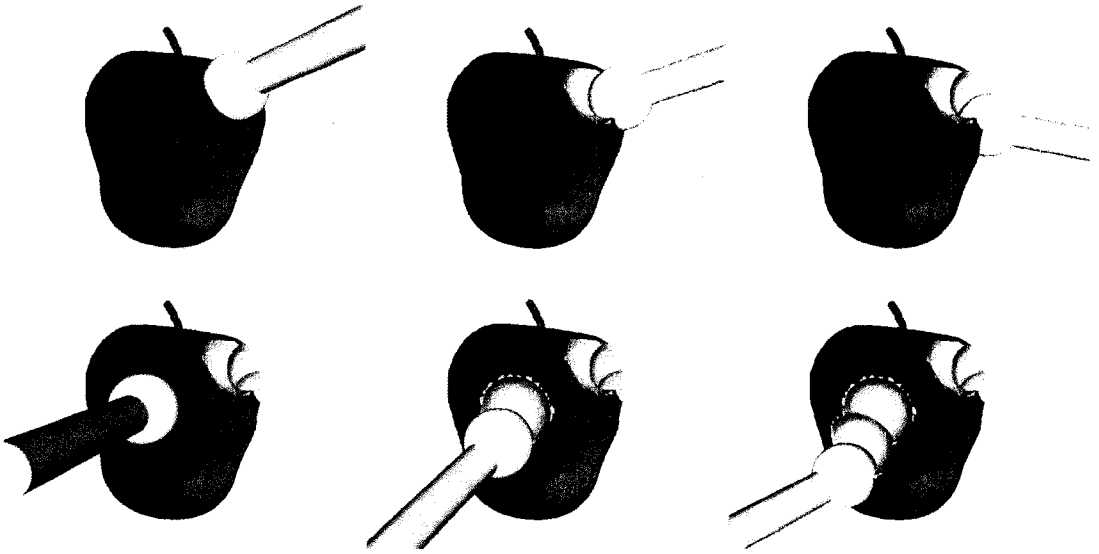


Figure 28: Demonstration of carving an externally textured object. The apple uses an external 2D texture and a simple white internal colour.

Qualitative results of our carving system are shown in Figures 27 and 28. In the figures carving is performed as discrete cuts rather than through continuous motion. In the textured figures 27 and 28, there are some artifacts that appear around the edges of the cuts. This is due to the removal of triangles that straddle the boundary of the cut that are then regenerated using the internal texture.

4.2. Performance

The performance of our method depends on the size of each individual cut performed when carving. As the number of voxels removed in a cut increases so does the amount of work needed to remesh the corresponding hole in the mesh, since the size of the boundary of the cut and the number of triangles to be generated is proportional to the number of new surface voxels.

To analyze the performance of the system we performed cuts on a hip model using a tool with a spherical carving head. The hip model was discretized into a 120x120x120 voxel set. Our experiments were performed on a 2.8 GHz Pentium D

processor machine. The OpenSceneGraph graphics library was used for rendering. Each cut was performed using a sphere centred on the hip surface.

Voxels Cut	Create Front (ms)	Remeshing (ms)	Total (ms)	Frames / Second
42.8	6.7	5.1	11.8	84.7
148.8	23.4	15.9	39.3	25.4
382.3	26.8	38.4	65.2	15.3
680.3	44.2	59.1	103.3	9.7

Table 2: Carving performance test results

The performance results are shown in Table 2. Each row in the table is the average of ten sample cuts using the carving tool. The size of the cutting head was increased every ten cuts so that the number of voxels removed each cut increased. *Voxels Cut* is the average number of voxels removed per cut. *Create Front* is the average time in milliseconds to remove the voxels and mesh triangles, and compute the new front. *Remeshing* is the average time to re-mesh the front using DBPA. *Total* is the sum of *Create Front* and *Remeshing*. *Frames / Second* is the upper bound on the number of frames per second that can be achieved based on the time taken to cut the voxels, computed as $1000 \times Total^{-1}$.

Figure 29 shows the total time to perform a cut taken from Table 2 as a graph.

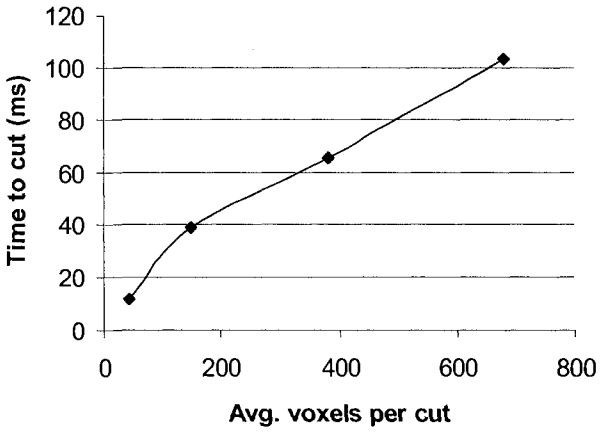


Figure 29: Graph of average time to complete a cut of varying numbers of voxels

Based on the results our system can achieve real-time rates on a large data set when removing below about 148 voxels per cut, which is approximately a volume of 5x5x5 voxels removed each cut.

To put this in perspective, as of today the general resolution of individual CT scan images is 512 pixels by 512 pixels, taking into account computational complexity and system cost [Pointer 2008]. At this resolution the shaft of the femur, having a diameter of approximately 23.4mm [Wikipedia 2008], can be represented with each voxel being 0.045mm in length if the image plane is perpendicular to the axis of the bone. This means we can simulate removal of up to 5.7 mm3 of volume 25 times per second, or 143 mm3 per second in real time.

The graph in Figure 29 demonstrates that though it does take longer to regenerate the triangle mesh after each cut as the number of voxels affected by the cut increases, the time taken increases generally linearly. In Table 2 we see that the time taken to generate a new front increases slower than the time taken to re-mesh the front. This is intuitive since generating the new front logically involves processing less surface area than generating

the new triangles for the cut area; in the worst case an area roughly equal to the diameter of the carving tool would be processed in the former, while in the latter an area roughly equal to the outer surface area of the tool head would be processed.

4.3. Validation

In order to measure the accuracy of the carving system we designed an experiment in which the volume of the cuts performed by our system could be compared to the actual volume that should have been cut. A large sphere was cut by a smaller sphere, with the smaller sphere's centre set on the surface of the larger sphere. For each cut, two values were compared and the relative error measured between the actual and measured results:

1. the first value is the amount of volume removed by the cut, and
2. the second value is the amount of volume remaining in the object being cut after the cut

We refer to value 1 as $V_{cut,actual}$ for the actual value, and $V_{cut,measured}$ for the measured value. We refer to value 2 as $V_{remain,actual}$ and $V_{remain,measured}$ for the actual and measured remaining volumes respectively.

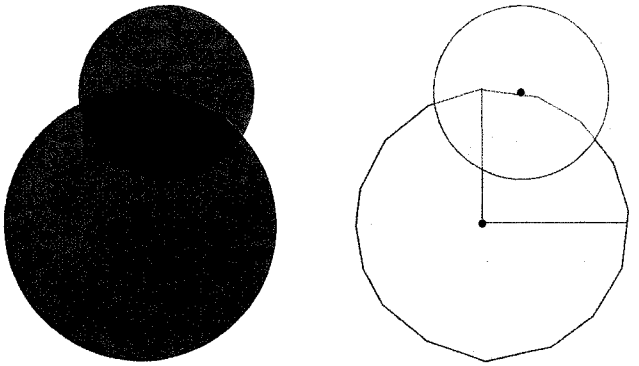
For the actual volume, $V_{cut,actual}$ was measured as the analytically computed value of the intersection of the sphere being cut W_{actual} and the sphere doing the cutting T_{actual}. The equations for computing the intersecting portions of the sphere were the equations for spherical caps. The value $V_{remain,actual}$ was simply the original sphere volume less $V_{cut,actual}$.

For the measured volume, the sphere being cut was converted to voxels and then converted to a mesh using our system. This sphere was then cut using the cutting sphere.

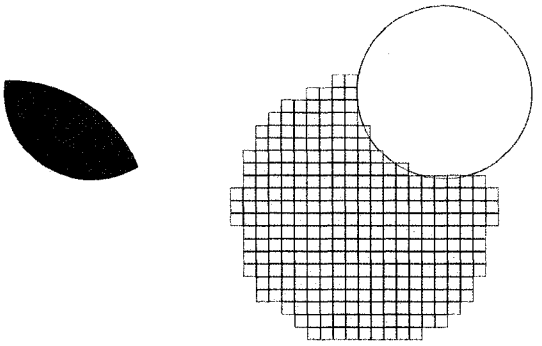
The value $V_{remain,measured}$ could then be computed as the volume of the remaining mesh after the cut was performed. The value $V_{cut,measured}$ is computed as the volume of the original sphere's mesh less $V_{remain,measured}$.

Since the voxels are arranged in a grid structure, the orientation of the cutting sphere with respect to the grid would cause variations in the accuracy of the cuts. For this reason the experiments were performed such that a series of cuts were performed over an octant of the sphere being cut. With the centre of the cutting sphere placed at the origin then translated along the Y axis for a distance equal to the radius of the sphere being cut, the cutting sphere was rotated about the X axis and then Z axis for all combinations of rotation values between PI/2 radians in increments of PI/40 radians (4.5 degrees). This is a total of 400 measurements. The spheres are recreated before each cut (the cuts are not cumulative).

Since the accuracy of the cuts intuitively depends on the resolution of the voxel space used to represent the sphere being cut for the measured values, the above measurements were repeated for different voxel resolutions to gauge the effect. Figure 30 demonstrates the validation concept.



(a)



(b)

Figure 30: Method for validation. In (a) the intersection of two spheres will be computed analytically (left), and empirically using our method with a mesh sphere over voxels and a spherical cutting tool (right), shown here as 2D circles. The computation is performed a number of times with the centre of the smaller sphere touching the larger at different positions within an octant of the larger sphere, shown here as a quadrant of the circle. In (b), the analytical intersection (left) will be compared to the result achieved by cutting voxels using our method for each position of the carving sphere.

Graphs of the results of these measurements are shown in Figure 31 and Figure 32. For the tests the sphere being cut had radius 10 and the cutting sphere had radius 3. In the graphs, the percentage error was calculated as:

$$\left|\frac{measured - real}{real}\right| \tag{20}$$

The graph in Figure 31 shows the average percentage error of 400 cuts for different voxel resolutions of the measured remaining volume to the actual remaining volume. The voxel resolutions represent the resolution along each axis of the voxel volume, so 100 for example denotes a voxel space of 100x100x100 voxels. The standard deviation in the percentage error does not appear in the chart since it is very low, of the magnitude of 0.01 % error.

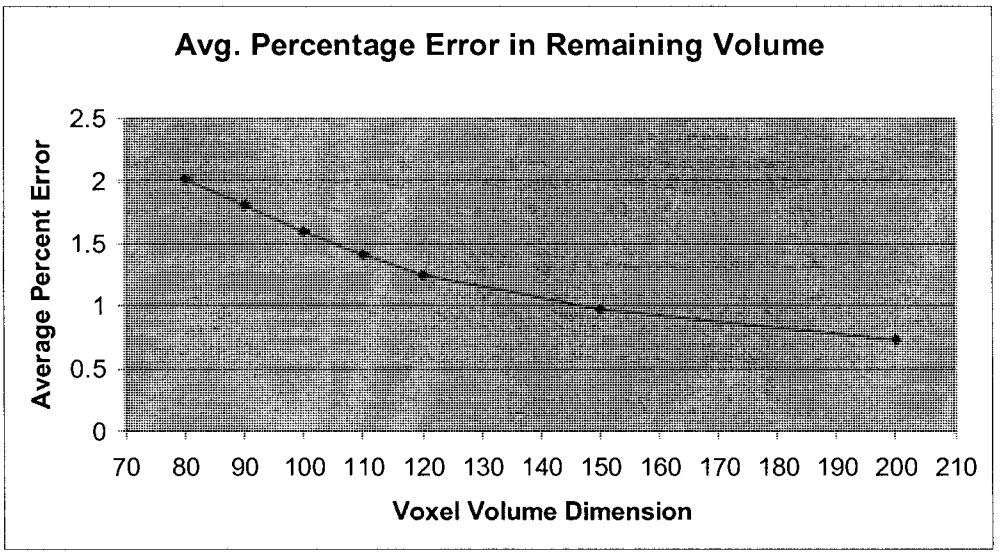


Figure 31: Accuracy comparison of system versus actual for the remaining volume after carving. Each data point is the average of 400 cuts about an octant of a sphere.

From the graph it is clear that the percentage error in the remaining volume is low compared to the original volume. The error is 2% for an 80x80x80 voxel resolution, and improves as the dimension of the voxel space increases.

The graph in figure 32 shows the average percentage error of 400 cuts for different voxel resolutions of the measured volume removed to the actual volume removed. The standard deviation in the percentage error is shown as error bars.

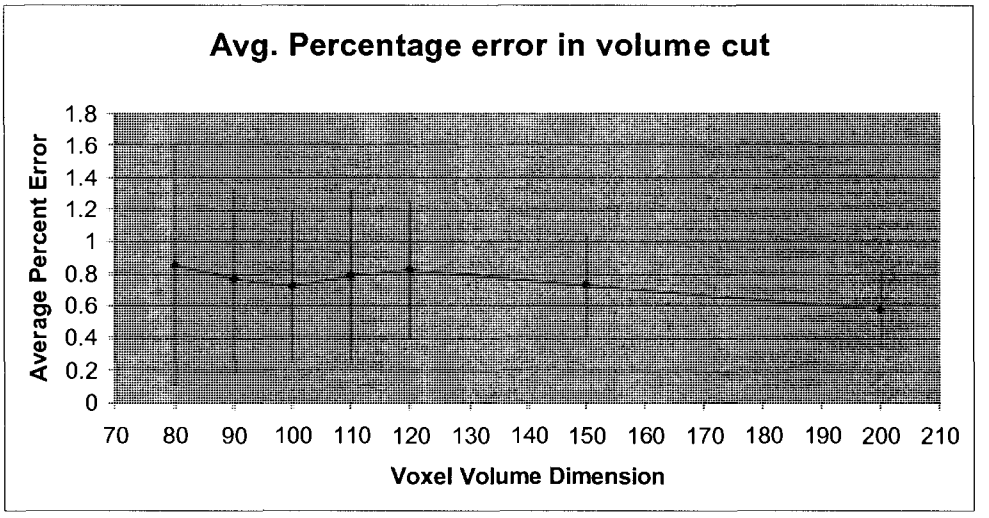


Figure 32: Accuracy comparison of system versus actual for the volume removed by carving. Each data point is the average of 400 cuts about an octant of a sphere. Error bars show the standard deviation.

The graph shows a much higher variation in the average percentage error since the difference between the actual and measured values are being related to a smaller volume. Overall the error is almost constant for different voxel resolutions, though the standard deviation seems to increase, showing that the cuts having with the worst error at lower resolutions have less bad error at higher resolutions.

In order to more closely analyze the affect of the voxel orientation on the carving accuracy we have partitioned the data from the previous graphs into four bands of different rotation of the cutting sphere centre about the z-axis: all measurements between 0 to 22.5 degrees of rotation from the x-axis in the octant, between 22.5 and 45 degrees, between 45 and 67.5, and finally between 67.5 and 90 degrees. The bands are shown as different series in Figure 33 and Figure 34.

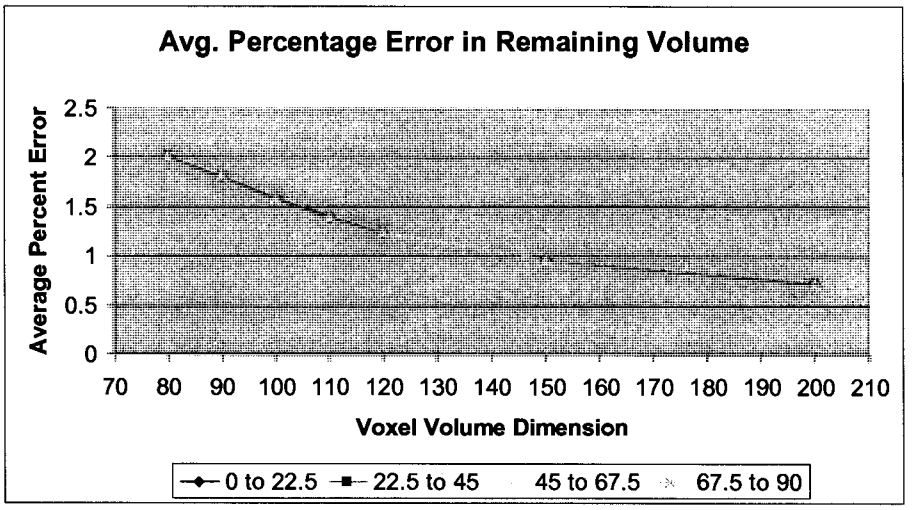


Figure 33: Banded accuracy comparison of system versus actual for the remaining volume after carving. Each data point is the average of 400 cuts about an octant of a sphere. Each series represents data samples in which the angle of the cutting sphere centre from the origin with respect to the x-axis was within the specified number of degrees.

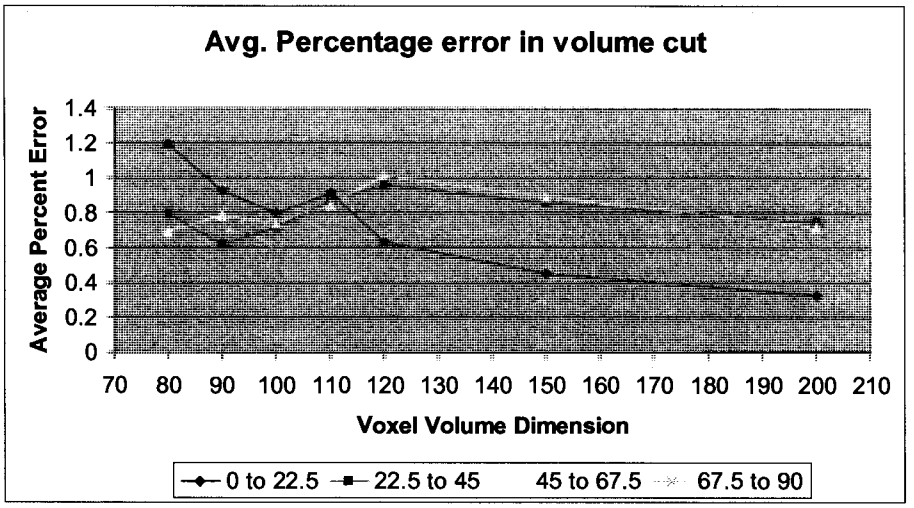


Figure 34: Banded accuracy comparison of system versus actual for the volume removed by carving. Each data point is the average of 400 cuts about an octant of a sphere. Each series represents data samples in which the angle of the cutting sphere centre from the origin with respect to the x-axis was within the specified number of degrees.

As seen in Figure 33 there is little variation in accuracy when comparing the actual remaining volume to the volume calculated by our method. However in Figure 34 the accuracy of the bands differs more significantly since a smaller volume is being represented with the same voxel space resolution as the larger volume, and so slight differences in the triangulation are more apparent. The graph shows that the bands in which the direction of the centre of the cutting sphere from the origin is nearly orthogonal to the voxel faces (bands 0 to 22.5 and 67.5 to 90) have slightly less percentage error on average for higher resolutions.

Chapter 5 Conclusion

In the preceding two chapters presented our approach that provides a visually realistic simulation of the effect of removing rigid bounded volumetric portions of a 3D object. By using a dual representation of the object being cut as both voxels and a mesh we can achieve both efficient volume removal by operating on the voxels, and a pleasing visualization by rendering the mesh. The conversion between the voxel representation and mesh representation is done using our novel DBPA algorithm which can re-triangulate selected portions of a mesh based on removal from the underlying point set.

Efficient visualization of the changing object shape is achieved by using voxels as an underlying volume representation and only updating the areas of the object mesh that were affected by the removal of voxels using the novel DBPA algorithm. Our method supports texturing using both 3D and 2D textures simultaneously for added realism.

We have shown that the performance of the method is dependent on the number of voxels removed during the individual cuts, and that for well-sized cuts our method can carve at sufficient rates for realistic real-time rendering. Through experimentation we have empirically shown that our method introduces minimal error into the object during carving.

5.1. Contributions

The following contributions were made as a result of performing the research described in this thesis:

1. Developed a scheme for combining a voxel representation of an object for shape changing operations with a mesh representation for visualization that for the first time has empirical results demonstrating the accuracy of the method.
2. Devised the Dynamic Ball-Pivoting Algorithm, an extension of the classic BPA algorithm that allows localized changes to the BPA point cloud after the BPA has already completed with localized reconstruction. The DBPA is what enables the interactive carving system to operate.
3. Derived and presented clearly the solid geometry calculations needed to implement the BPA that were lacking in the original paper
4. Improved the performance of the BPA when it is applied to points arranged as a 3D grid
5. Demonstrated the accuracy and performance of our hybrid voxel/mesh approach

5.2. Discussion

Our approach achieves real-time rates, but further optimization might be possible. In our re-meshing algorithm, when we are creating front cycles we apply the BPA glue operator for each edge of the front that is found to be coincident at that time. It is theoretically possible to instead determine connected paths of coincident edges and to perform an extended glue operation on the coincident paths instead of each edge. In some cases this could improve performance since in two cases of glue one cycle of edges must be copied into another cycle. However this extended operation is difficult to implement, and introduces new special cases to handle.

Though not shown in our figures, any possible shape besides a sphere can be used as a cutting head as long as it is possible to determine whether points are inside or outside

of the shape. Shapes that can be represented using implicit equations are particularly well suited.

Our method has a limitation inherited from the BPA. If a voxel is removed and the only valid point for re-triangulating from the new front is already part of the mesh (i.e. from the opposite side of the shape) then the front is left as a set of boundary edges. This can leave a one-sided hole in the shape, or can cause the back-side of the faces already using the voxels to be visible within the cycle of boundary edges. Generally this will only occur when a cut has removed voxels such that a one-voxel plane is left. For simulation purposes one work-around is to detect when this will occur, and refuse to remove those voxels in order to maintain a consistent boundary representation.

Though we have only used the DBPA algorithm to update a display mesh due to removal of points in the underlying point set, it should be straightforward to apply the algorithm to the addition of points to the underlying point set. From a high-level this would involve keeping track of the set of surface voxels that become occluded by the addition of new volume and using this set to generate the rim of edges from which the new triangles will be generated.

5.3. Future Work

As visible in the figures in Chapter 3 section 8, when a 2D external texture is applied and the object is cut there can be some visual artifacts round the edge of the cut, being particularly visible when the internal texture contrasts with the external. This is caused by the removal of voxels near the edge of the cut creating new triangles that are close to planar with the surface of the object yet textured with the internal texture. Finding a solution would greatly improve the visual accuracy of the method.

For very large cuts that remove a large number of voxels our method does not achieve real-time rates. To improve the animation when performing these large cuts one solution might be to divide the cut into a number of smaller cuts which can be performed faster and then perform each of the smaller cuts. Though it would still take at least as much time to perform the larger cut as before, the user would have more visual feedback and so the system would seem more responsive. It would be ideal if the cut could be quickly divided into layers that follow the contour of the cutting tool surface as this would arguably appear more realistic.

Our approach is ideally suited to being controlled by a haptic input device as it allows intuitive interactive manipulation. An ideal solution would provide a more physically realistic approach to the removal of volume as compared to simple volume intersection and provide force feedback to the user. Gabriel Telles O'Neill has developed a system for haptic control of our cutting method as a fourth year honours project.

References

[Acosta 2007] Eric Acosta, Alan Liu. "Real-time volumetric haptic and visual burr hole simulation". *IEEE Virtual Reality*, 2007, pp. 247-250

[Agus 2003] Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A, Picasso B, Sellari Franceschini S. "A haptic model of a bone-cutting burr". *Stud Health Technol Inform.* 2003; 94:4-10.

[Akenine-Möller 2005] Tomas Akenine-Möller. "Fast 3d triangle-box overlap testing". *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 8, New York, NY, USA, 2005. ACM.

[Akkouche 2001] S. Akkouche and E. Galin. "Adaptive implicit surface polygonization using marching triangles". In *Computer Graphics Forum*, volume 20(2), pages 67–80, 2001.

[Ayasse 2003] Ayasse, J. "Discrete Displacement Fields: A Versatile Representation of Geometry for Simulation in Computer- Aided Manufacturing". 2003.

[Bernardini 1999] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. "The ball-pivoting algorithm for surface reconstruction". *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[Bloomenthal 1997] J. Bloomenthal, C. Bajaj, M.P. Cani-Gascuel, A. Rockwood, B. Wyvill and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, 1997.

[Boltcheva 2007] Dobrina Boltcheva, Dominique Bechmann, and Sylvain Thery. "Discrete delaunay: Boundary extraction from voxel objects". In *3-D Digital Imaging and Modeling, 2007: Sixth International Conference on.* 2007.

[Bourke 1997] P. Bourke. "Implicit surfaces". http://ozviz.wasp.uwa.edu.au/~pbourke/modelling_rendering/implicitsurf/, 1997.

[Cermak 2004] M. Cermak and V. Skala. "Adaptive edge spinning algorithm for polygonization of implicit surfaces". In *Computer Graphics International*, volume 16-19, pages 36–43, 2004.

[Chen 2004] Yisong Chen and Horace Ho-Shing Ip. "Texture evolution: 3d texture synthesis from single 2d growable texture pattern". *The Visual Computer*, 20(10):650–664, 2004.

[Crossno 1999] Patricia Crossno and Edward Angel. "Spiraling edge: fast surface reconstruction from partially organized sample points". In *VIS '99: Proceedings of the conference on Visualization '99*, pages 317–324, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[Ebert 1998] Ebert, D. S., Worley, S., Musgrave, F. K., Peachey, D., Perlin, K., and Musgrave, K. F. 1998 Texturing and Modeling. 2nd. Academic Press, Inc.

[Engel 2004] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron E. Lefohn, Christof Rezk Salama, and Daniel Weiskopf. "Real-Time Volume Graphics". *SIGGRAPH Course Notes 28*, 2004

[Fang 1998] S. Fang and R. Srinivasan. "Volumetric-csg – a model-based volume visualization approach", 1998.

[Forest 2002] C. Forest, Herve Delingette, and Nicholas Ayache. "Cutting simulation of manifold volumetric meshes". In *MICCAI '02: Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II*, pages 235–244, London, UK, 2002. Springer-Verlag.

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

[Gopi 2002] M. Gopi and Shankar Krishnan. "A fast and efficient projection-based approach for surface reconstruction". In *SIBGRAPI '02: Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing*, pages 179–186, Washington, DC, USA, 2002. IEEE Computer Society.

[Hadwiger 2002] Markus Hadwiger, Joe M. Kniss, Klaus Engel and Christof Rezk-Salama. "High-Quality Volume Graphics on Consumer PC Hardware". *SIGGRAPH Course Notes 42*, 2002.

[Hart 1996] John C. Hart. "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces". In *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[Hilton 1997] Hilton and J. Illingworth. "Marching triangles: Delaunay implicit surface triangulation", 1997.

[Hua 2004] Jing Hua, Hong Qin, "Haptics-Based Dynamic Implicit Solid Modeling". *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 5, pp. 574–586, Sept/Oct, 2004.

[Ito 2004] Yasushi Ito, Alan M. Shih and Bharat K. Soni. "Reliable Isotropic Tetrahedral Mesh Generation Based on an Advancing Front Method". *Proceedings, 13th International Meshing Roundtable*, Williamsburg, VA, Sandia National Laboratories, pp.95-106, September 19-22 2004

[Keren 1998] D. Keren and C. Gotsman. "Tight fitting of convex polyhedral shapes". *International Journal of Shape Modeling*, pages 111–126, 1998.

[Kopf 2007] Kopf, J., Fu, C., Cohen-Or, D., Deussen, O., Lischinski, D., and Wong, T. 2007. Solid texture synthesis from 2D exemplars. *ACM Trans. Graph.* 26, 3 (Jul. 2007), 2.

[Liao 2002] Duoduo Liao and Shiaofen Fang. "Fast volumetric csg modeling using standard graphics system". In *SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 204–211, New York, NY, USA, 2002. ACM.

[Loop 2006] Charles Loop and Jim Blinn. "Real-Time GPU Rendering of Piecewise Algebraic Surfaces". In *SIGGRAPH Proceedings*, Boston, 2006

[Lorensen 1987] William E. Lorensen and Harvey E. Cline. Marching cubes: "A high resolution 3d surface construction algorithm". In *Computer Graphics (Proceedings of SIGGRAPH '87)*, volume 21-4, pages 163–169, 1987.

[Morris 2004] D. Morris, C. Sewell, N. Blevins, F. Barbagli, and K Salisbury. "A collaborative virtual environment for the simulation of temporal bone surgery". In *Proceedings of MICCAI*. Springer-Verlag, 2004.

[Morris 2005] D. Morris, S. Girod, F. Barbagli, and K Salisbury. "An interactive simulation environment for craniofacial surgical procedures". In *Proceedings of MMVR (Medicine Meets Virtual Reality)*, 2005.

[Muller 1997] Heinrich Muller and Michael Wehle. "Visualization of implicit surfaces using adaptive tetrahedrizations". *Scientific Visualization Conference*, 1997.

[Naylor 1990] Bruce Naylor, John Amanatides, and William Thibault. "Merging bsp trees yields polyhedral set operations". *SIGGRAPH Comput. Graph.*, 24(4):115–124, 1990.

[Overveld 2004] Kees van Overveld and B. Wyvill. "Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface". *The Visual Computer*, 20-6:362–379, 2004.

[Owada 2004] Shigeru Owada, Frank Nielsen, Makoto Okabe, and Takeo Igarashi. "Volumetric illustration: designing 3d models with internal textures". In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 322–328, New York, NY, USA, 2004. ACM.

[Pham 2000] Pham DL, Xu C, Prince JL. "A survey of current methods in medical image segmentation". *Annu Rev Biomed Eng: Annu Rev 2000*; 2:315-37

[Picinbono 2001] G. Picinbono, H. Delingette, and N. Ayache. "Nonlinear and anisotropic elastic soft tissue models for medical simulation". In *Proceedings of IEEE International Conference on Robotics and Automation,* 2001, pages 1370–1375, 2001.

[Pointer 2008] David Pointer. "Computed Tomography (CT) Scan Image Reconstruction on the SRC-7". In *Proceedings of the Fourth Annual Reconfigurable Systems Summer Institute (RSSI'08)*, 2008.

[Raviv 1999] Raviv A. and G. Elber, "Three Dimensional Freeform Sculpting via Zero Sets of Scalar Trivariate Functions," *Proc. Fifth ACM Symp. Solid Modeling and Applications*, pp. 246–257, 1999.

[Ren 2006] Ren, X. y.; Mueller and H.; Kuhlenkoetter, B., "Surfel-based surface modeling for robotic belt grinding simulation", *Journal – Zhejiang University Science A*. Vol. 7, no. 7, pp. 1215–1224, 2006.

[Rossignac 1999] J. Rossignac and A. Requicha. "Solid modeling". In J. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons, 1999.

[Schmidt 1999] "Development of a Time-Resolved Optical Tomography System for Neonatal Brain Imaging". Florian E. W. Schmidt, (PhD dissertation, University College London, 1999)

[Sela 2004] Guy Sela, Sagi Schein, and Gershon Elber. "Real-time incision simulation using discontinuous free form deformation". Lecture *Notes in Computer Science*, 3078/2004:114–123, 2004.

[Seland 2007] Johan Seland and Tor Dokken. "Real-Time Algebraic Surface Visualization". In *Geometric Modelling, Numerical Simulation, and Optimization*, pp. 163-183, 2007.

[Slater 2002] Mel Slater, Yiorgos Chrysanthou, and Anthony Steed. *Computer Graphics And Virtual Environments: From Realism to Real-Time*. Addison-Wesley, 2002.

[Taubin 1993] Gabriel Taubin. "An improved algorithm for algebraic curve and surface fitting". In *Fourth International Conference on Computer Vision (ICCV '93)*, pages 658–665, Berlin, Germany, May 1993.

[Thibault 1987] William C. Thibault and Bruce F. Naylor. "Set operations on polyhedra using binary space partitioning trees". *SIGGRAPH Comput. Graph.*, 21(4):153–162, 1987.

[Turk 2002] Turk, G. and O'brien, J. F. "Modelling with implicit surfaces that interpolate" *ACM Trans. Graph.* 21, 4 (Oct. 2002), 855-873.

[Wikipedia 2005] Wikipedia. "CSG Tree", http://en.wikipedia.org/wiki/Image:Csg_tree.png, Aug. 2005

[Wikipedia 2008] Wikipedia. "Femur", http://en.wikipedia.org/wiki/Femur, Nov. 2008

[Zwicker 2001] Zwicker, M., Pfister, H., van Baar, J., and Gross, M., "Surface Splatting", *ACM SIGGRAPH*, ISBN: 1-58113-374-X, pp. 371-378, August 2001