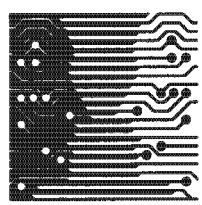
TOPOLOGY ALTERATION OF MESHES USING DIRECTLY MANIPULATED FREE-FORM DEFORMATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Barry Steyn
June 2004

Supervised by
James E Gain



Abstract

The field of virtual sculpting aims to create computer models by mimicking an artist's studio and materials on computer. There are many ways to implement virtual sculpting. One method is to use a class of transformations known as the spatial deformations. Spatial deformations alter the spatial position of an object's primitives, resulting in warping. This warping effect is controlled to produce features of an object, thereby allowing an artist to 'sculpt' an object. While an object's primitives are altered, its connective structure is left intact. This limits spatial deformation to warpings that do not change topology, since it is necessary to change geometry for topology alteration.

This dissertation attempts to produce an interactive method for topology alteration by using a spatial deformation known as Direct Manipulation of Free-Form Deformation (DMFFD). Since DMFFD is a spatial deformation, it is impossible to use it by itself to alter topology. However, a process developed for animation can be used in conjunction with DMFFD to alter topology: A n-dimensional object is extruded to (n+1) dimensions, and then deformed. From the deformed extruded object, it is possible to extract a topologically altered original object.

This dissertation attempts to adapt this process to virtual sculpting by making it interactive. A 3D object is extruded into 4D and given tetrahedral connection information. This tetrahedronalisation allows an object to be unambiguously extracted from the extruded object. 3D DMFFD is adapted to 4D to deform the extruded object. Extrusion is performed by intersecting a 3D hyper-plane with the extruded object to obtain a point set which is then triangulated. Optimisations are developed for the extrusion, deformation and extraction phases in order to make the process interactive. Models of various sizes are tested to prove interactivity.

List of Tables

List of Figures

Chapter 1

Introduction

Computer graphics has grown quickly, becoming an important field in computer science. Exploited in a diverse range of fields, whole industries have been created around it (e.g., cinematic effects and scientific visualisation). It essentially involves the display of discrete data on a computer, but the acquisition of this data has become a problem necessitating active research. One aspect of the data acquisition problem is object creation, with the current standard being to digitise clay macquettes [14].

Many modelling packages have been created that seek to replace the digitising of clay macquettes. In many instances, the artist must know how the computer scientist has designed the modelling package in order to properly use it [35]. Virtual sculpting was invented to allow object creation without prior computer science knowledge being required. Its pioneers made the comparison of an artist being able to sculpt with a clay-like medium [51] thereby lessening the learning curve for a designer. To this end, anything that is possible in clay should be possible in virtual sculpting, including:

- **Object Separation**: The ability to tear a piece off an object.
- **Hole Creation**: The ability to create holes in an object.
- **Object Joining**: The ability to join objects.

The above items are examples of topology alterations. If virtual sculpting is to provide a viable alternative to other methods these operations must be available.

Current graphics hardware is optimised to accept objects represented by a polygonal mesh. In essence, a polygon mesh consists of two properties [23] (Figure 1.1):

- **Point Set**: Points in space (known as vertices) defining the object's boundary.
- **Connection Information**: Information concerned with how individual vertices are connected to define a surface.

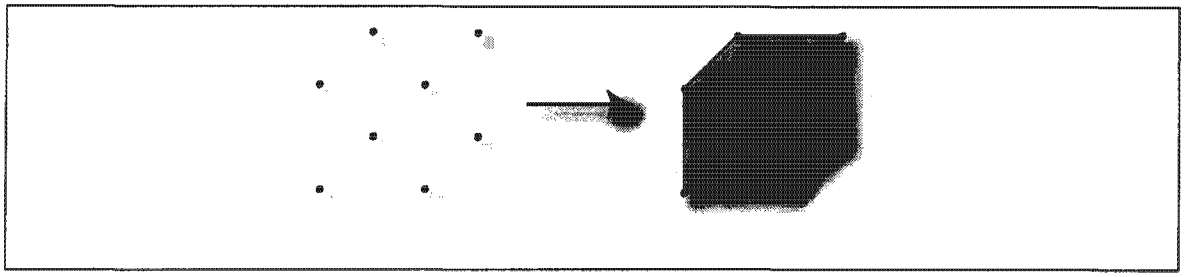


Figure 1.1: **Connectivity Information**: A point set is given connectivity information in order to attain structure.

If an object is to undergo a topology alteration, it is necessary for the connectivity information to be altered. For example, if a sphere is to become a torus, it needs a hole through the middle of it. To create a hole, the connectivity information of the sphere must change.

1.1 Virtual Sculpting

Virtual sculpting aims to provide an environment for the creation of computerised objects in a manner that resembles traditional sculpting. It is therefore vital that any virtual sculpting scheme be interactive. It is a relatively old field, with Parent introducing the topic in 1977 [51] by using a decay function to implement it. As the field developed, better means than decay functions have been used for implementation [46]. This dissertation uses virtual sculpting implemented with a form of spatial deformation based on work done by Gain [26].

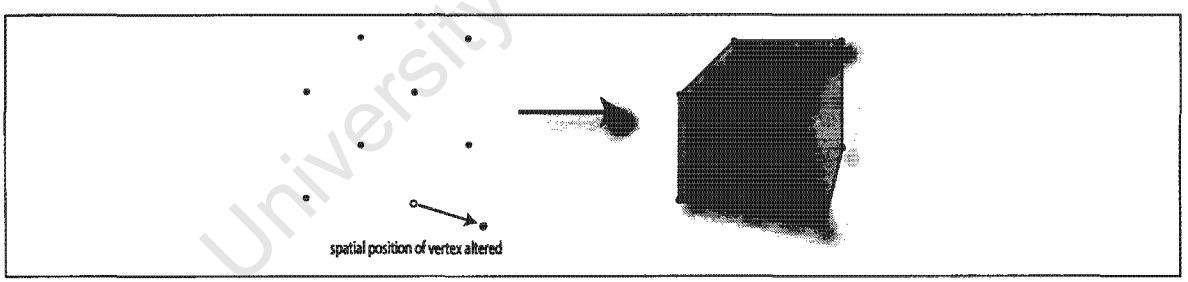


Figure 1.2: **Spatial deformation**: Objects are deformed by altering the spatial positions of their vertices.

1.1.1 Spatial Deformation For Virtual Sculpting

Spatial deformation achieves a warping effect by altering the spatial position of an objects vertices (Figure 1.2). One of its attractions is that it is independent of object representation. Therefore connection information is not altered under spatial deformation.

Free-Form (FFD) deformation, introduced by Sederberg and Parry [55] is an instance of spatial deformation. FFD has the advantage that it can be used interactively on relatively complex objects [6].

It is also able to simulate a clay medium relatively well, making it easier for an artist to perform sculpting. A detraction of FFD is that an artist cannot directly manipulate an object to deform it. This poses a usability problem as object sculpting becomes less intuitive.

Direct Manipulation of Free-Form Deformations (DMFFD) alters FFD by allowing an artist to directly modify an object while being efficient enough to be interactive on relatively complex meshes. It is the chosen method for the implementation of virtual sculpting concerning this work.

1.1.2 Topology Alteration Under Virtual Sculpting

Topology alteration should adhere to the following properties:

- **Interactivity**: Interactive feedback is crucial to virtual sculpting. It must therefore also be present in any scheme that attempts to change the topology of an object under the guise of virtual sculpting.
- **Validity**: The object must conform to certain standards of mathematical correctness and the external realism of the object should not be compromised by the topology change process. The former implies that C_1 continuity be present and the latter implies that artifacts such as self intersection be eliminated.
- **Multifariousness**: Intentions by the artist must be modelled with ease.

Spatial deformation deforms an object by changing the spatial positions of its vertices. But for a topology change to take place, connectivity of the object must change: It is **not** enough to alter the spatial position of the vertices. Therefore, it is impossible to change the topology of an object using spatial deformations alone.

1.2 Dissertaion

This dissertation is focused on providing a scheme for altering an object's topology under virtual sculpting, implemented with DMFFD.

Problem Description

To alter an objects topology using DMFFD whilst adhering to *interactivity* and *validity*

The multifariousness property is considered to be in the realms of usability, and as such is not considered in this dissertation.

Dissertaion Overview

This dissertation is presented as follows:

- Chapter 2 **Prior Art**: A brief review is presented of past and current techniques used to alter the topology of an object under the guise of modelling. These techniques are compared with the properties of section 1.1.2

- Chapter 3 **Foundations**: Necessary background information is established for understanding this dissertation. An in depth discussion of virtual sculpting is given, as well as a formal presentation of FFD, DMFFD and topology. A structure, developed for producing animation, known as a space-time object, is presented and motivated for use. Space-time objects can be divided into three parts; *extrusion, deformation* and *extraction.* The disadvantages of space-time objects in virtual sculpting are discussed.

- Chapter 4 **Extrusion**: The extrusion part of the space-time object is enhanced for virtual sculpting. A three dimensional (3D) object is extruded into four dimensions (4D). An efficient means for creating 4D face and edge information is discussed. Various enhancements are made by exploiting coherence which enables extrusion speed up.

- Chapter 5 **Deformation**: A formal mathematical presentation of FFD and DMFFD in 3D is presented. DMFFD is then extended into four dimensions. A discussion of the interactivity of DMFFD in 3D, and hence in 4D is presented.

- Chapter 6 **Extraction**: An unambiguous interactive extraction is presented. Extraction involves intersecting the 4D object with a 3D hyper-plane, thus obtaining a point set in 3D. The point set is then triangulated, resulting in an object that is potentially topologically altered. The topology change is reliant upon the deformation of the 4D object.

- Chapter 7 **Results and Evaluation**: Timing results are shown demonstrating that optimisations in this dissertation enable interactivity.

- Chapter 8 **Conclusion and Future Work**: Interactivity is achieved with the optimisations. A test for self intersection as well as a refinement/decimation scheme are subjects for future work.

Chapter 2

Literature Review

Various methods for altering an objects topology are presented and evaluated in this chapter.

This chapter is partitioned as follows:

- **Section 2.1 - Topology Alteration Techniques**: Various techniques to change the topology of a model are examined. There are three fields that are concentrated upon:
  - **Section 2.1.1 - Volumetric Modelling**: An object is modelled using 3D cells called voxels, resulting in the internals being represented.
  - **Section 2.1.2 - Implicit Surface**: Objects are represented using an implicit function.
  - **Section 2.1.4 - Boundary Representation Modelling**: An infinitely thing boundary representing a surface is used to model the object.
- **Section 2.2 - Evaluation**: Topology alteration techniques are evaluated.
- **Section 2.3 - Summary**

2.1 Topology Alteration Techniques

2.1.1 Volumetric Modelling

Volumetric structures explicitly represent an object's volume using 3D cells. Current rendering hardware requires that volumetric objects be converted to polygonal meshes, and this is most commonly done by using marching cubes [42] or some variant of it. Volumes provide an intuitive means to represent complex topologies. They are also easy to edit interactively and therefore are appropriate for model creation. The internal volumetric data structure has many applications, for example, medical imaging or the representation of diffuse materials such as clouds [17], but is unnecessary

for modelling the appearance of solid objects [2], since the internals seldom need rendering. An infinitely thin skin that models the surface of a solid object will suffice (this field of modelling is called boundary representation). This in conjunction with higher overhead costs compared to boundary representations is the reason why volumetric data structures are not standard.

Volumetric Sculpting

Influenced by 2D paint programs, volumetric sculpting involves editing a voxmap, analogous to a 2D bitmap, but composed of voxels [38]. Voxels are the 3D equivalent of pixels. Whereas pixels can be visualised as a square with equal length and breadth, voxels are a cube with equal length, breadth and height. Voxels are flagged to represent material or space, with volumetric sculpting providing an interface with which these flags are altered. Thus material can be created, or removed.

Volumetric sculpting was first introduced by Galyean and Hughes [29]. Voxalised objects are sculpted using tools which are also Voxalised. Voxels are axis aligned in a static voxmap lattice thus incurring aliasing problems when an object's surface lies at strange angles relative to the major axis. Galyean and Hughes attempt to solve the problem by using a low-pass filter to sample both the tool and the object. While the modelling technique is free-form, it cannot achieve fine detail due to the relatively sparse sampling of the voxels. To increase efficiency, the voxalised object is polygonised using an incremental marching cubes algorithm that is only performed on the modified region.

Wang and Kaufman [59] extend volumetric sculpting to allow several objects in the world space to be sculpted independently. Each object to be sculpted has its own coordinate system within world space, with sculpting only affecting the local coordinates of the object.

Ferley *et al.* [21, 22] eliminate the restrictive use of a static voxmap while still maintaining interactivity. Voxels are composed of eight corners, each corner with a flag that represents whether it is in material or space. Corners and voxels are stored in balanced binary trees, respectively, with voxels that are on the boundary between space and material being stored in a separate binary tree. The user is able to add corners at any moment, thereby providing an unlimited voxmap size. Interactivity is achieved by rapidly determining the boundary voxels, which is accomplished by examining the corners that are changed due to tool interactions.

Agarwala [2] generalises the voxel approach by the use of slabs. Like Ferley *et al.*, slabs consist of eight corners, but differ by having a coordinate system and not being constrained to a cuboid shape. Therefore the sides need not be parallel. The local coordinate systems of the slab are aligned to the surfaces they represent, providing an extraction that is not aliased compared to voxels.

Distance Fields

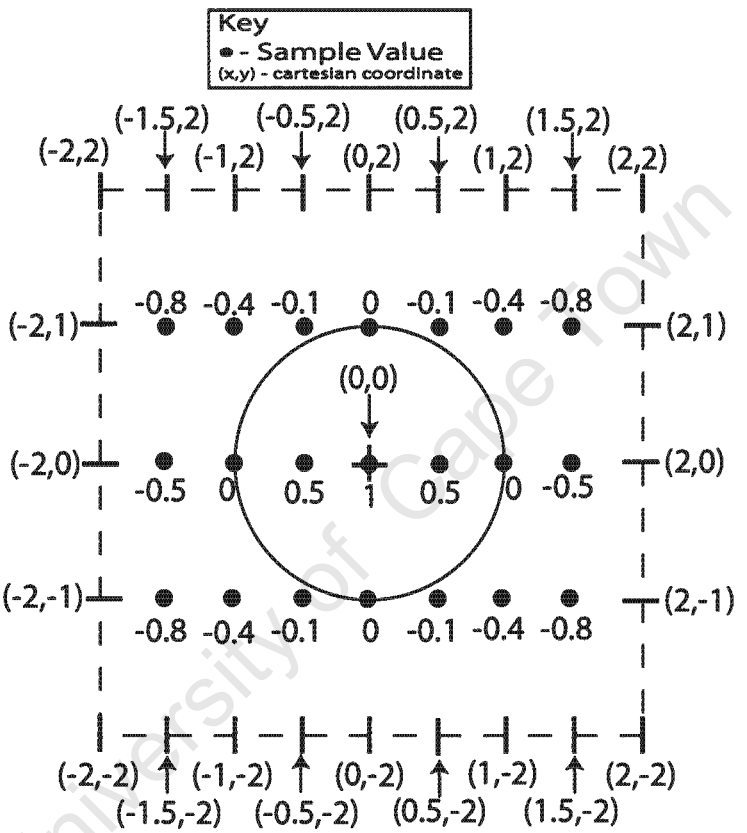
A distance field is a regularly sampled scalar field representing the minimum distance to an object. Distance is calculated with any pre-defined metric, and if it is signed, can represent the exterior and interior of an object (Example 2.1). Distance fields have many applications besides computer

Example 2.1 - Distance Field Of Unit Sphere

Any metric will suffice to calculate distance. The unit sphere described by $\sqrt{x^2 + y^2} = 1$ can be represented by the following distance functions:

- **Euclidean Signed Distance:** $f(x, y) = 1 - \sqrt{x^2 + y^2}$
- **Algebraic Signed Distance:** $f(x, y) = 1 - \sqrt{x^2 + y^2}$
- **Unsigned Distance:** $f(x, y) = (1 - (x^2 + y^2))^2$

A regularly sampled distance map for the Euclidean signed distance could be as follows:



graphics, such as path planning for robotics [40] and the generation of swept volumes [54].

Gibson [31] proposed sampled distance fields for volume rendering. She proved that while volumetric models have advantages over surface models when objects have complex topologies, they represent objects built from intensity based data badly e.g. data from CT scans. Distance fields were shown as a viable alternative. Surfaces can be easily generated from the level set (zero value) of the distance field, while surface normals could be reconstructed from the distance field gradient. Distance fields also allow for the representation of objects with complex topologies.

A variant of the distance field, the adaptively sampled distance field (ADF) [24] was suggested by Perry and Frisken. A problem with regular distance fields is that dense sampling is required for areas of high curvature. An object that has local portions of high curvature will have a regularly

Example 2.2 - Producing A Sphere Using An Implicit Surface

The function:

$$f(x, y, z) = \sqrt{(x_o - x)^2 + (y_o - y)^2 + (z_o - z)^2} - 3$$

Defines a sphere with radius 3 and center (x_o, y_o, z_o)

sampled dense distance field imposed upon it, even though dense samplings are redundant for the portions of low curvature. ADF's solve this problem by adaptively subdividing the distance field at high curvature. The result is a distance field that requires less space than traditional distance fields with good curvature approximation.

Kizamu [52] was created as a virtual sculpting tool to exploit ADFs. Both the object and tools are embedded in the ADF. Sculpting is implemented by applying CSG operations to the object and the tool distance fields. There are two types of operations: *differencing* and *additive*. Topology alteration can be achieved by using a tool for differencing with the object for example using a differencing tool with a sphere to make a donut. It can also be achieved by using an additive tool, for example, a tool that adds a handle to a mug.

2.1.2 Implicit Surface

Set theoretic operations are the basis of Constructive Solid Geometry (CSG). CSG uses operations in conjunction with primitives to produce more elaborate shapes [23] including topology alterations (Example 2.3). CSG Primitives are usually represented by polygonal models.

Implicit surfaces generalise CSG but with primitives being represented by an implicit function and set theoretic operators defined as part of the function [56]. Introduced by Blinn [9] and refined by Wyvill et al. [62], implicit surfaces are objects composed of isosurfaces determined from a 3D scalar field [10]. They are defined by the following equation:

$$f(P) = c,\ P \in \mathbb{R}^3, c \in \mathbb{R} \tag{2.1}$$

P is an arbitrary point in space, and $f(P)$ is the level-set value associated with it, usually proportional to the distance between P and the intended surface. Points in space are in one of three states according to their implicit value:

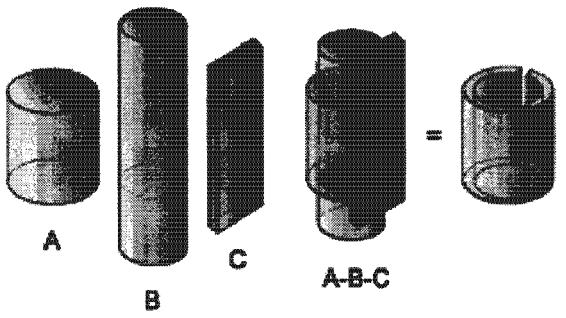
- $\mathbf{f(P) = 0}$: P is on the surface.
- $\mathbf{f(P) < 0}$: P is on one side of the surface (nominally the inside)
- $\mathbf{f(P) > 0}$: P is on the other side of the surface.

Level Set Modelling

As the name suggests (see above), level set models are defined as iso-surfaces of an implicit surface. Museth et al. [48] present level set functions, a speed function applied to the iso-surface that

Example 2.3 - Topology Alteration Using CSG

Assume three primitives: A small fat cylinder, a long thin cylinder, and a thin . The union of these primitives results in a conglomeration of the shapes, but differencing them alters the topology of the small fat cylinder:



are able to blend, smooth and emboss the surface. It has the benefits of producing smooth non self intersecting models that can easily have its topology altered (such as surface joining and hole creation). The volumetric nature of it implies an overhead in computation time and memory. But techinques developed limit computations to a narrow band around the level set of interest [1] making computational complexity proportional to the surface area of the model. This limits the process to models that do not have very large surface areas. Also, due to the implicit nature of the surface being modelled, a volumetric representation becomes necessary. Conversion between this representation to a B-Rep (Section 2.1.4)is also an overhead.

Constructive Solid Geometry

Constructive Solid Geometry (CSG) uses simple solid shapes (primitives) that can be easily defined to build more complex shapes using boolean operators [47]. The boolean operators are *union*, *difference* and *intersection*, and can easily change an objects topology (Example 2.3). The resulting model is stored in a tree, with leaf nodes being primitives, and branches being boolean operators.

Skeletons

Implicit surfaces based on skeletons [43] are defined by a set of skeletal elements $s_i (i = 1 \ldots n)$ each with a set of associated field functions f_i:

$$f(P) = \sum_{i=1}^{n} w_i f_i(d(P, s_i))$$

where $d(P, s_i)$ is the distance metric (which is not necessarily a Euclidean metric [58]) from P to s_i, and w_i is a weighting coefficient. Skeletons can be any geometric surface; points, lines, curves, etc. The blending properties of implicit surfaces mean smooth surfaces are created by skeletal interaction. An object's topology is altered by skeletal interaction, for example, a hole can be created in a sphere by intersecting it with a cylinder biased with a negative weight.

Volumetric Implicit Surfaces

Volumetric implicit surfaces are voxalised objects with voxel density values obtained using an implicit function. This approach combines both the advantages of volumetric representation and implicit surfaces. Implicit objects are usually polygonised to take advantage of graphics hardware, and this is generally done using a marching cubes [42] approach.

Hua and Qin [36, 37] use trivariate B-Splines [20, 23] to define an implicit function. The trivariate B-Spline functions define several hyper-patches in 3D space. Assume there are N B-Spline patches over a sculpting workspace, each with an arbitrary location and orientation, the implicit function is defined as follows:

$$f(P) = \sum_{i=1}^{N} s_i(\mathbf{T}_i(P))$$

s is the generic trivariate B-Spline functions with transform $\mathbf{T}$ embedding points from 3D ($\mathbb{R}^3$) to the parametric domain of s. The object represented by the implicit trivariate B-Spline function is discretised into voxels, each containing a scalar value sampled at a grid point derived from the implicit function.

2.1.3 Subdivision Modelling

Pioneered by Catmull and Clark [13], subdivision modelling defines a smooth surface or curve as the limit of successive refinements:

$$G = \lim_{i \to n} G_{i+1} = F(G_i), \qquad n, i \in \mathbb{N}^+$$

where G_0 is the initial shape, and F is the subdivision transformation. The closer n becomes to infinity, the more accurate the subdivision surface becomes. F consists of two distinct phases [60]:

- **Splitting**: An edge or face is split into two edges or four faces respectively
- **Averaging**: A new vertex introduced to the model is positioned at a fixed affine combination of its neighbour's position.

A classic example of subdivision models are fractals [5]. Subdivision's popularity is due to its simplicity and ease of use.

McDonnell and Qin [44, 45] base their sculpting tool on Catmull-Clark subdivision solids. Subdivision solids consist of an initial control lattice that is recursively subdivided, thereby refining the space encompassed by the lattice. McDonnell and Qin use a modified radial-edge data structure [49] to represent the solid. This data structure allows one to efficiently determine component neighbours and simplices. Topology is changed and updated relatively easily due to this data structure. Parts of the solid can be easily removed to change genus, and resubdivided to produce a smooth model.

2.1.4 Boundary Representation Modelling

Boundary Representation (B-Rep) models consist of interconnected polygons that form a surface mesh [23]. The mesh describes an infinitely thin hollow shell (a surface) around the interface of the object, and, therefore, B-Rep modelling has also been referred to as surface modelling in the literature [2]. B-Rep modelling is currently the standard means of object representation due to mainstream graphics hardware being optimised for rasterisation between polygon meshes. It is for this reason that other modelling representations usually convert to B-Rep. Since surfaces do not model volume, altering the topology is not trivial.

Surface meshes are not easy to create or edit, with a multitude of modelling packages designed to this end (such as 3D Studio Max and Maya). These packages are notorious for being difficult to use, making laser scanning a clay maquette [53, 14, 41] the current industry standard (e.g. for films).

The field of spatial deformation [6, 46, 30] (Section 3.1.1) was developed to provide an intuitive and powerful means for creating B-Rep models. It essentially repositions points in space, which in turn warps the mesh. A user would start with an easily defined base mesh (e.g. a sphere) and through spatial deformation create more complex models. Active research in the field has produced spatial deformation techniques that are both powerful and easy to use [26].

Higher Dimensional Deformation

It is impossible for objects to undergo a topology alteration with spatial deformation alone (see Section 3.1.1 for an explanation). But a topology alteration can be attained [12, 4] by extruding the object into an extra dimension, spatially deforming the extruded object, and extracting a possibly topologically altered object compared to the original. Topology alteration relies on two things:

- **Deformation**: The type of deformation that is performed on the extruded object.
- **Extraction**: The place in the extruded dimension chosen to extract the object.

The correct deformation and extraction can be derived from experimentation, therefore it is a viable and predictable method.

2.2 Evaluation

Since volumetric sculpting represents an object's internals, hole creation can be easily accomplished. Additive operations such as creating an arch on a surface can also be performed with ease. But the discrete nature of voxels make it necessary for a large number of voxels to be present in order to avoid aliasing. Voxalised objects must also be converted to B-Rep models via a marching cube scheme in order to exploit current graphics hardware. It would be difficult to achieve interactivity and a fine level of detail for large objects using a volumetric approach.

Regularly sampled distance fields also suffer from aliasing: A distance field must be finely sampled to represent high curvature. Fine sampling is wasted on low curvature, therefore regularly sampled distance fields have a large overhead in terms of space. This is effectively overcome by adaptively sampling the distance field resulting in fine sampling for high curvature and sparse sampling for low curvature.

CSG is limited to constructing complex objects using primitives, resulting in a lack of 'blobiness' that is easily produced using implicit surfaces. Complex topology can easily be represented, but operations to change topology are limited to boolean operations with other CSG objects, therefore it suffers from the lack of smoothness inherent in all objects built using CSG.

Skeletons produce smooth 'blobby' objects due to the properties of implicit surfaces. But it is difficult to produce sharp edges e.g., a crease in a surface. Topology alteration and the representation of complex topologies are inherent to this method. A B-Rep model is produced by determining the level-set of the implicit surface (the zero value of the implicit function) and can be converted to a B-Rep format by using a voxel based partitioning schemed with marching cubes. This could potentially degrade interactivity for large data. Skeletal interaction is often non-intuitive, with alteration of the implicit function's coefficients leading to unexpected changes [36].

Volumetric implicit surfaces have demonstrated promise, but the method presented in [36] is not able to produce sharp creases. It is well suited as a means to represent and alter complex topology, combining the best features of implicit functions and volumetric sculpting. Inevitably, it also has some of its undesirable features, such as having to convert to a B-Rep format using a marching cubes like algorithm.

Subdivision surfaces defer the topology alteration problem to the base mesh, where the connection information must be changed. The more iterations applied to the base-mesh, the smoother the surface becomes. This may potentially have a negative impact on interactivity. Subdivision surfaces does not lend itself to the virtual sculpting metaphor of warping a clay-like substance.

Higher dimensional deformation has the added cost of having to extrude an *n-dimensional* object to an *(n+1)-dimensional* object, and then after a deformation, extract it to produce a topology alteration. In applications designed to exploit higher dimensional deformation such as animation [4], interactivity is not necessary. It is also not able to perform object joining unless the object has been previously split. But the deformation algorithm for 4D can be adapted from code and concepts produced in [26], and it also has the added advantage that it can be applied directly to B-Rep models, as well as producing topology alterations that conforms to the properties specified in Chapter 1. Adapting higher dimensional deformation to be interactive would make this method, apart from not being able to perform object joining, ideal for topology alteration under virtual sculpting. Higher dimensional deformation is the chosen method for topology alteration for this dissertation. Overcoming its pitfalls will be the aim of this dissertation.

	Advantages	Disadvantages
Volumetric Sculpting	• Can represent complex topology. • Topology can be altered easily.	• Current graphics hardware not optimised for volumetric representation. • Incurs aliasing problems.
Level Set Modelling	• Complex topology alterations are possible. • Resulting model is smooth.	• Model cannot be too big. • Overhead in terms of B-Rep creation.
Distance Fields	• Complex topology can easily be represented and altered.	• Limited resolution unless adaptively sampled.
Constructive Solid Geometry	• Topology can easily be changed. • Powerful method for object modelling.	• Complex shapes limited to unions of primitives. • Topology alterations do not produce smooth surfaces.
Skeletons	• Topology can be easily and smoothly changed by skeletal interaction.	• Difficult to model objects with sharp changes in continuity.
Volumetric Implicit Surfaces	• Topology can easily be changed. • Powerful method for object modelling.	• Suffers from same problem as implicit surfaces and volumetric sculpting.
Subdivision Modelling	• Topology alteration is C_1 continuous and smooth.	• Does not lend itself to virtual sculpting.
Higher Dimensional Deformation	• Conforms to properties of topology alteration introduced in Chapter 1. • Works for B-Rep.	• Does not lend itself to object joining. • Not interactive.

2.3 Summary

Current topology alteration techniques in the literature have been examined, and evaluated. The techniques evaluated were volumetric sculpting, distance fields, skeletons, constructive solid geometry, volumetric implicit surfaces, subdivision modelling and higher dimensional deformation. A motivation for choosing higher dimensional deformation was presented in the conclusion of the evaluation section.

Chapter 3

Foundations

In this chapter, the necessary background information needed to understand the remainder of this dissertation is established, in particular virtual sculpting, topology alteration and a data structure that is able to alter an object's topology, namely the space-time object.

This chapter is partitioned as follows:

- **Section 3.1**: A detailed introduction to virtual sculpting is presented, highlighting properties that are essential for all implementations. Details concerning the chosen method for implementation are then discussed:
  - **Section 3.1.1**: Spatial deformations are given a mathematical formalism, after which their disadvantages and advantages are discussed.
  - **Section 3.1.2**: A high level explanation of free-form deformation is presented, with a discussion concerning its strengths and weaknesses.
  - **Section 3.1.3**: An overview of direct manipulations of free-form deformations is expounded with reasons as to why it is a better option than free-form deformation.
- **Section 3.2**: Mathematical details of topology in an intuitive setting are given.
- **Section 3.3**: Aubert and Bechmans' [4] space-time object is presented, with emphasis on its topology altering abilities. This method is motivated for use under virtual sculpting and, in addition, each of the three stages comprising a space-time object, namely extrusion, deformation and extraction are discussed.
  - **Section 3.3.1**: The shortcomings of space-time objects under virtual sculpting are analysed.
- **Section 3.4 - Summary**

3.1 Virtual Sculpting

A common problem in the field of computer graphics is the acquisition of three dimensional (3D) models. One approach is to triangulate a point cloud acquired by laser scanning a clay maquette [53, 14, 41]. It is the current industry standard due to its relatively accurate depiction of the input clay maquette and because it allows the designer to work in a familiar clay medium.

Despite its relative accuracy, laser scanning has many undesirable properties: It is expensive, time consuming, has limited resolution, produces unnecessarily large data-sets with artefacts [32, 15] and is unable to digitise occluded or hard-to-reach surfaces. Furthermore, after the artist surrenders the maquette to the laser scanner, his/her job is done. What appears on the computer may be slightly different to what was envisaged. Although laser scanning has many disadvantages, it is still the standard method of 3D model acquisition, a case in point being the special effects industry [16].

Virtual sculpting is a computational modelling technique that emulates traditional sculpting [8]. It offers several advantages over the costly and cumbersome laser scanning of clay models: models can be edited without needing to resculpt, the artist controls the entire creation cycle and artifacts due to laser scanning are eliminated. If virtual sculpting is to resemble traditional sculpting, an implementation should possess the following:

- **Interactivity** - In a physical medium, such as clay, changes occur instantaneously. Virtual sculpting should therefore provide interactive feedback[1].
- **Flexibility** - It should be possible to replicate the full range of shapes possible under conventional sculpting.
- **Aesthetics** - The final object should appear smooth, exhibiting C_1 continuity[2]. Physical properties such as volume preservation and gravity can be ignored.
- **Correctness** - The model must be geometrically correct and thus should not self-intersect, since this looks aesthetically unappealing and may cause some renderers to fail.

There are a variety of manners in which virtual sculpting can be implemented [46, 30, 6]. This work uses virtual sculpting as implemented by Gain [25, 26]. Gain adapts a form of *spatial deformation* for virtual sculpting.

3.1.1 Spatial Deformation

Spatial deformations are a collection of non-physical[3] deformable models that are efficient enough to be interactive and powerful enough to provide a means for virtual sculpting. In essence, spatial deformations alter the spatial positions of points in space.

[1] Generally accepted to be fifteen updates-per-second.
[2] Existence and continuity of first derivatives.
[3] Non-physical meaning it is not based upon Newtonian physics

A spatial deformation is performed by embedding an object from world space into a deformable space, and then warping the deformable space. The warping effect of the deformable space is carried through to the embedded object, after which the deformed object is extracted into world space.

The initial embedding from world space object coordinates, $\mathcal{X} = (x, y, z)$, to coordinates of the deformable space $\mathcal{U} = (s, t, u)$, can be described formally as a mapping $\mathcal{D} : \mathbb{R}^3 \rightarrow \mathbb{R}^n$. The dimension of the deformable space is algorithm specific and thus cannot be assumed. Analogously, the extracting of an object from the deformable space to world space can be described by the mapping $\mathcal{E} : \mathbb{R}^n \rightarrow \mathbb{R}^3$. These two mappings are used to define the following equations:

$$\mathcal{E}(\mathcal{X}) = U \tag{3.1}$$

$$\mathcal{D}(\mathcal{U}) = \tilde{\mathcal{X}} \tag{3.2}$$

Equation 3.1 is known as the embedding equation and is used to embed objects into deformation space. Equation 3.2 is known as the deformation equation, as it maps coordinates from deformation space to world space. When the mappings $\mathcal{D}$ and $\mathcal{E}$ are composed, a mapping $\mathcal{F} : \mathbb{R}^3 \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^3$ that maps undeformed world space coordinates to deformed world space coordinates is produced. This mapping defines spatial deformation:

$$\mathcal{F}(\mathcal{X}) = \mathcal{D} \circ \mathcal{E}(\mathcal{X}) = \mathcal{D}(\mathcal{U}) = \tilde{\mathcal{X}} \tag{3.3}$$

Spatial deformation is therefore a composition of the embedding equation with the deformation equation. It is vital that these equations be efficient if spatial deformation is to be interactive.

Limitations of spatial deformations

In a virtual sculpting paradigm, spatial deformations have the following limitations:

- **Mesh Degradation**: This is due to the underlying representation of polygonal meshes (Section 1). Meshes use vertices as primitives. Linearly connected vertices are able to model straight lines perfectly, but are not so suited to modelling curvature. It is necessary that a large amount of closely spaced vertices be present to approximate high curvature. Spatial deformation alters the spatial position of vertices, potentially making vertices that were close together far apart, thereby degrading curvature. This problem can be solved by using an adaptive refinement scheme, that adaptively inserts vertices as soon as curvature degrades.
- **Topological Invariance**: It is necessary that the relationships governing the vertices, edges and faces (i.e. the connectivity information) change in order to alter topology [57]. Spatial deformations only alter the spatial position of points leaving the connectivity information constant. Thus it is impossible to alter the topology of an object using spatial deformations alone.

Topological invariance is not as straightforward to solve as the mesh degradation problem. This dissertation addresses this problem.

3.1.2 Free-Form Deformation

Free-form deformation (FFD), introduced by Sederberg and Parry [55], is a spatial deformation technique able to provide a warping mechanism that resembles traditional sculpting. A deformable space is set up by arranging points, known as control points, in a regular cuboid lattice structure. Object vertices are then embedded in the lattice structure, and are evaluated as a weighted sum of the control points. The deformable space, known as a hyper-patch, is warped by altering the spatial position of the control points. Warping is carried through to the object vertices that are now evaluated as a weighted sum of the warped control points. Sedarberg and Parry describe this process as embedding an object in a maleable substance such as jelly, and then warping this substance to warp the embedded object (Figure 3.1).

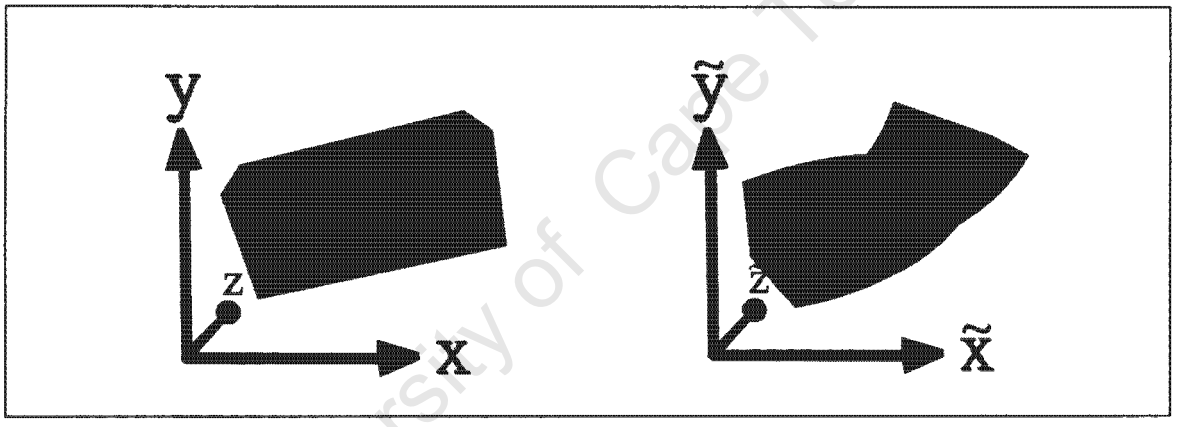


Figure 3.1: **The Free-Form Deformation Jelly Metaphor**: [Left] An object is embedded in a maleable jelly like substance from world space coordinates [Right] The jelly substance is warped, with the warping effect carried through to the embedded object. The object is now in 'warped' world space coordinates.

Deformation is performed by altering control points with no input required from the embedded object. FFD is therefore considered to indirectly warp the object. This poses usability problems because deformations become less intuitive. Even worse is if a control point that needs altering is occluded by the embedded object. FFD has the following properties:

- **Local Deformation**: Imposing a hyper-patch over a small area of the object or by using using a B-Spline basis allows for deformation over a local portion of the object [20].
- **Volume Destruction**: In contrast to real sculpting, FFD does not preserve volume. It is possible to enforce volume preservation [55], but this could possibly reduce cognitive overhead since object shapes may change more drastically than intended under deformation.

- **Interactivity**: FFD is efficient enough to be interactive; one of the required properties of virtual sculpting and topology alteration.

3.1.3 Direct Manipulation Of Free-Form Deformation

Direct manipulation of free-form deformation [35] attempts to solve the usability issues of FFD by allowing the user to specify deformations directly on the object. Given a FFD deformable space, DMFFD functions by calculating the spatial reconfiguration of lattice control points to comply with a user specified warping of the object. After which, the FFD is process is used to warp the object. Since the user is not required to alter the lattice, control points are hidden. The heart of the DMFFD algorithm is calculating a pseudo inverse for a matrix of B-Spline basis weights evaluated for each control point (see Section 5.1.2 for a detailed explanation). This is inefficient and has potential to violate the interactive property needed for virtual sculpting.

3.1.4 Virtual Sculpting As Implemented By Gain

This dissertation uses DMFFD as improved and implemented by Gain [26, 25] as a means for virtual sculpting. Gain has enhanced DMFFD for virtual sculpting fixing many of the problems inherent to spatial deformations, FFD and DMFFD. The following features were implemented:

- **Adaptive Refinement And Decimation**: An efficient refinement scheme that adds extra vertices to areas of curvature that get degraded under deformation was implemented. Alternatively, a decimation scheme that removes vertices when curvature sampling is deemed to fine was also implemented [27].
- **Efficient Pseudo-Inverse Calculation**: DMFFD pseudo-inverse calculation was improved beyond the method presented by Hsu et al [35]. Improvements allow interactive calculation of the pseudo-inverse [25].
- **Efficient Self-Intersection Test**: A *sufficient* self intersection test that is efficient was developed. A *necessary* self intersection test was also developed, but due to its inefficiencies, it degrades interactivity [28]

All the above features do not degrade the interactivity requirement of virtual sculpting unduly.

3.2 Topology

The topology of an object is one of many factors that determines the relationship between an objects vertices, edges and faces[4] (i.e. the *connectivity* of the object). In 1750, Euler [3] produced his famous

[4]Other factors being curvature, planarity etc.

formula for the geometry of a simply connected (genus 0) polyhedra:

$$v - e + f = 2 \tag{3.4}$$

This relates the number of vertices (v), edges (e) and faces (f) in a polyhedron. In 1813, the mathematician Lhüilier observed that for solids with holes, this connectivity formula did not hold. Instead, a polyhedral solid, with n holes obeyed the following formula:

$$v - e + f = 2 - 2n \tag{3.5}$$

The number of holes in an object determines its *genus*: An object of genus n has n holes. Equation 3.5 is a direct consequence of an object's topology: Objects with different topologies must ***necessarily*** have different connectivity pertaining to their geometry. Objects with identical topology can have matching connectivity.

3.2.1 An Intuitive Description Of Topology

A Topology defines a relation between two spaces which are the same if a homeomorphic function[5] exists between the two spaces. In determining if two spaces differ topologically, it is not possible to test the infinite number of mappings between them and conclude that none of them are homeomorphic. Instead, mathematicians prove spaces are topologically distinct by determining a relationship pertaining to one of the space's topology that does not hold for the other space. These relationships are called topological invariants, and as the name implies, they are constant for spaces with the same topology. For spaces consisting of polyhedra with genus 0, Euler's equation (equation 3.4) is a topological invariant. Analogously, Lhülier's equation (equation 3.5) is a topological invariant for spaces consisting of polyhedra with holes. Thus these two spaces are not of the same topology.

Section 1 details topology alterations that are important under virtual sculpting.

The Donut And The Tea Cup: An Intuitive Test To Determine Topological Equivalence

There is a common mathematical joke that says a topologist cannot tell the difference between a donut and a tea cup. The reason is that a donut is topologically equivalent to a tea cup. This can be intuitively shown by the fact that a donut can be spatially deformed into a teacup without a connectivity change. An intuitive test as to whether one object is topologically equivalent to another is if the object can be ***warped*** into the other.

3.2.2 Simplicial Complex Objects

A simplicial complex $\mathcal{K}$ is a collection of simplices in $\mathbb{R}^n$ such that [34]:

1. Every element of a simplex $\mathcal{K}$ is itself a simplex in $\mathcal{K}$

[5] A function that is injective, surjective, continuous and has a continuous inverse.

2. The intersection of any two simplices in $\mathcal{K}$ is a face of each of them.

The first of the above points implies that simplices used to build an object are simplicial complexes themselves. The second point implies that the intersection of faces will be a simplex in $\mathcal{K}$. An N-Simplex is a simplex that conforms to the two above points in dimension N (See Figure 4.3).

3.2.3 Manifold Objects

A manifold X of a point $x \in \mathbb{R}^n$ is a set X of points that contains an open ball of radius $\epsilon > 0$, $\epsilon \in \mathbb{R}^n$ centered at x. A manifold object has a manifold set X for every point contained in it.

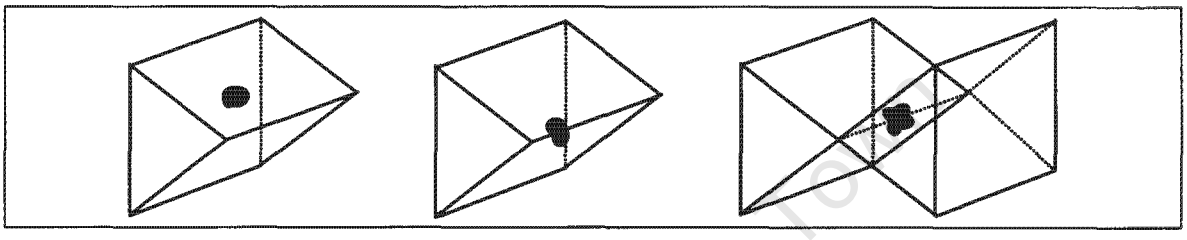


Figure 3.2: **2-Manifold And Non 2-Manifold Objects**:[Left and Middle] A 2-Manifold object: for every point, an open ball in $\mathbf{R}^2$ exists [Right] A non 2-Manifold object: There exists at least one point that does not have an open ball neighbourhood that exists in $\mathbb{R}^2$.

A 2-Manifold object has every point with a manifold whose open ball is in $\mathbb{R}^2$ - a disc. Polygon meshes that are 2-Manifold are 'well behaved' with no more than two faces sharing an edge (Figure 3.2.3).

Features such as texturing and neighbour calculation rely upon these restrictions, therefore 3D polygon meshes that are 2-Manifold (hence manifold) and simplicial complexes are used in this dissertation, and likewise, all objects produced are also 2-Manifold and simplicial complexes.

3.3 Space-Time Objects

Aubert and Bechman [4] proposed using a time extruded object in conjunction with spatial deformation for the purposes of animation. A time extruded object, called a space-time object, is an object that has been extruded into an extra dimension representing time. Therefore a 2D object produces a 3D space-time object, and a 3D object produces a 4D space-time object (Example 3.1). Extruding a n-dimensional object to a $n+1$-dimensional objects allows the extra dimension to represent time, with cross-sections representing the spatial position of the n-dimensional object at that time.

Assuming a space-time object has been linearly extruded and left undeformed, the original non-extruded object can be extracted from it by intersecting it with a hyper-plane.[6]

It is impossible for spatial deformation to alter the space-time object's topology, but after deformation, this assertion does not necessarily hold for the object extracted via hyper-plane intersection.

[6] A hyper-plane can be considered a generalisation of a plane existing in one dimension less than the current space. Therefore a hyper-plane in 3D space is a 2D plane and a hyper-plane in a 4D space is a 3D volume.

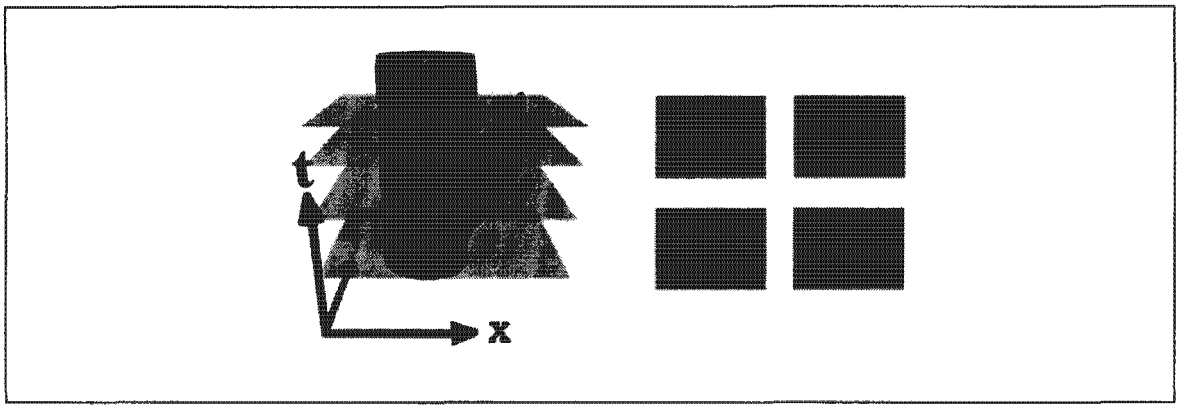


Figure 3.3: **Topology Alteration Of A Circle**: A circle is extruded into a cylinder after which it is deformed. Depending on the extraction, a separation will pertain to the circle.

Example 3.1 - A Three Dimensional Space-Time Object

A 3D space-time object is a two-dimensional (2D) object that has been extruded in 3D with the third dimension representing time. Assuming the 2D object was a disc, then the 3D extruded object would be a cylinder. If the disc was moving with a constant velocity, the cylinder would be slanted with a constant gradient towards its movement.

Thus a space-time object can be deformed in such a way that a topologically altered non-extruded object can be extracted from it.

A powerful spatial deformation tool called Deformation of Geometric Model Editor (DOGME) [11, 7] was developed with the ability to deforme space-time objects. DOGME in conjunction with space-time objects was aimed at producing a novel means for animation: A 3D object is extruded into 4D, with each instance of the fourth dimension representing the objects position during that time. Deformations made to the space-time object produce 'special effects' on the extracted object. This process is performed in three stages:

- **Extrusion**: A 3D object is extruded to produce a 4D space-time object.
- **Deformation**: The 4D object is deformed using a spatial deformation.
- **Extraction**: A 3D object is extracted from the 4D space-time object.

Each of the three stages will now be explained according to the implementation of 4D animation [4]:

Extrusion

Space-time objects are limited to manifolds[7] in order to make concrete the adjacency relations between vertices, edges and faces. Only vertices from the original model are embedded, with each vertex given by four components - (x, y, z, t) - where t is the time (fourth) dimension. Embedded faces and edges and their connectivity are deduced via linear interpolation of the embedded vertices.

[7]This is not a big limitation since most models available are already manifold.

Extrusions are not necessarily constant due to the fact that movement is modelled through time (Example 3.1).

Deformation

Deformation is performed using DOGME, a powerful general form of direct spatial deformation which allows a generalised mapping to and from multiple dimensions. Extrusion functions can be 'plugged' into DOGME in order to accomplish this mapping.

Extraction

The extraction algorithm is a three stage process:

- **Edge/Hyper-Plane Intersection**: Edges of the space-time object are intersected with the hyper-plane. This step provides an extracted point set.
- **Extracted Edge Creation**: Edges are created by connecting two vertices resulting from the intersection of the hyper-plane with a face of the space-time object.
- **Connectivity Information**: Connectivity information for the extracted object is deduced from the connectivity information of the space-time object.

Dogme's power is the ability to be abstract in terms of deformation. Since the deformation function and its dimension of operation is not specific, it is difficult to optimise it for topology alteration specifically. An interesting thing to watch out for is the development of deformation functions for DOGME specifically targeted to topology alteration. The method of Toplogy alteration used in this disseration 'copies' DOGME in extruded to a higher dimension to achieve topology change. But deformation is adapted specifically for topology alteration by using a form of Free-Form Deformation that is adapted for four dimensions.

3.3.1 An Analysis Of Space-Time Objects For Virtual Sculpting

For this method to be viable, it must conform to the requirements of topology alteration (Section 1.1.2) and virtual sculpting (Section 3.1). The following issues conflict:

- **Interactivity**: It is not vital that animation be performed in interactive time. It can instead be completed as a pre-process, and therefore space-time objects do not necessarily guarantee interactive rates.
- **Validity**: The validity of the model is compromised by the face structures in the space-time objects. The faces have the ability to become non-convex under deformation, and therefore produce ambiguous extractions (see Chapter 6 for more details).

This dissertation aims to address the above issues to make space-time objects viable under virtual sculpting.

The space-time object will provide topology alterations that are aesthetically pleasing in terms of continuity. It is for this reason that it has been chosen as the basis for topology alteration.

3.4 Summary

A detailed introduction to virtual sculpting was presented, with an explanation of spatial deformations - specifically, free-form deformation and direct manipulations of free-form deformations - as the means whereby virtual sculpting is implemented in this dissertation. Analysis of the shortcomings of spatial deformations (including both FFD and DMFFD), specifically shortcomings for topology alteration, was presented. An intuitive foundation into the mathematical field of topology was given, with emphasis on what a topology alteration with respect to a computerised model implies. The definition and advantages of a manifold model was explained using the notions of topology. A means of breaking the topology of an object by using a space-time object in conjunction with spatial deformation was introduced, along with reasons for and against using it for virtual sculpting in its current form.

Chapter 4

Extrusion

The extrusion process is discussed in this chapter. Extrusion is the first stage in the topology alteration process. Unlike [4], connection information is built into the extruded model. This is vital for an unambiguous extraction scheme (Chapter 6). The extrusion process is performed as part of the 3D model loading process. The user expects to wait a certain amount of time whilst the model loads, and it is during this time that the model is extruded. After an extraction, the object has to be extruded again. This can be performed in a thread whilst the user is examining the topology change. Extrusion therefore does not have to be interactive but must be made as fast as possible so as not to degrade the interactivity of the process as a whole.

The chapter is partitioned as follows:

- **Section 4.1 - The Extrusion Process**: A high-level overview of the extrusion process is presented, followed by an indepth explanation of each of its components:
- **Section 4.2 - Optimisations**: Optimisations are presented to improve extrusion performance time. These include:
  - **Section 4.2.1 - Structural Decisions To Decrease Memory Usage**: A 4D structure is much larger than its 3D counterpart. Decisions taken to decrease this structure are presented.
  - **Section 4.2.2 - Vertex Embedding Coherence**: Vertices are embedded with a linear coherence. Exploiting this coherence considerably improves extrusion time.
  - **Section 4.2.3 - Generalisation Of Structure**: The 4D structure can be generalised, so that only the first layer need be built, after which, all other layers can be extrapolated.
- **Section 4.3 - Summary**.

4.1 The Extrusion Process

The extrusion process produces a manifold 4D object from a 3D model. The 3D model is assumed to be a manifold triangle mesh containing only vertex and face information. A High-level overview of the process follows:

1. **3D Neighbour Creation**: Neighbour information is determined for the 3D model as a once off pre-process (Section 4.1.1).
2. **4D Point Set Creation**: Vertices from the 3D model are embedded in 4D within levels. Each level consists of all the vertices in the 3D model with a 4D offset unique to the level (Section 4.1.2).
3. **Connection Information**: The 4D point set is given connection information in the form of 4D prisms (Section 4.1.3).
4. **Subdividing Prisms Into Tetrahedra** - Prisms are subdivided into *tetrahedra* to yield planar and convex faces (Section 4.1.4).
5. **Edge Information** - Ordered edge information is produced for use in the *extraction* process (Section 4.1.5).

4.1.1 3D Neighbour Creation

Neighbour information for the 3D model is created as a once off pre-process. In order to conserve space and increase efficiency, neighbour information is not created for the 4D model. Neighbour traversal in 4D therefore relies upon neighbour information in the 3D model. 3D neighbour calculation is performed as follows:

- **Vertex Ordering**: The three vertex indices for each face are numerically ordered in ascending order. Assume these ordered indices are v_0, v_1 and v_2.
- **Edge Creation**: Three *ordered* edges, that reference the face they originate from are created from the ordered vertices. They are: $e_0 = (v_0, v_1)$, $e_1 = (v_0, v_2)$ and $e_2 = (v_1, v_2)$.
- **Edge Sorting**: The edge storage structure is sorted by imposing a lexical ordering on it. The ordering is as follows:
  $e_1 = (v_0, v_1) < e_2 = (v_2, v_3) \Rightarrow (v_0 < v_2)$ or $(v_0 = v_2$ and $v_1 < v_3)$
- **Neighbour Creation**: Adjacent entries in the storage structure are examined. If they are identical, the faces they reference are neighbours.

The above algorithm stores neighbour information in an implicitly ordered fashion: For a particular face, the first item in the face neighbour structure (face neighbour 1) will correspond to the face's

least edge with respect to the lexical ordering, the second item (face neighbour 2) the second least edge and the third item (face neighbour 3) the largest edge index (Example 4.1). This fact is exploited for extraction optimisation.

The data structure for 3D objects consists of vertices and triangle connectivity information. Each triangle has three references to its three triangle edge neighbours that are built using the above procedure.

4.1.2 4D Point Set Creation

Each Vertex (x, y, z) from the 3D model is embedded in 4D by appending an additional component to it (x, y, z, t). This is an iterative process; each iteration embedding all the vertices of the 3D model with a different 4D offset.

The number of iterations defines the number of *levels*, whilst the 4D offset defines the *distance* between levels. There is a fine balance between these two variables: Small offset values produce unnecessarily elaborate objects, whilst big values produce sparse objects. Conversely, a small number of levels produces a sparse object, whilst a large number of levels produces too intricate an object. Experimentation is needed to choose the correct number of levels and the distance between them. A sparse object results in extractions that may sample curvature poorly after deformation, whilst a too detailed object will produce objects with unnecessary density. For efficiency sake, it is desirable to produce the least number of levels possible, by decreasing the number of iterations. A refinement scheme may be necessary to ensure extractions do not degrade curvature (Chapter 8).

4.1.3 Creating Connection Information

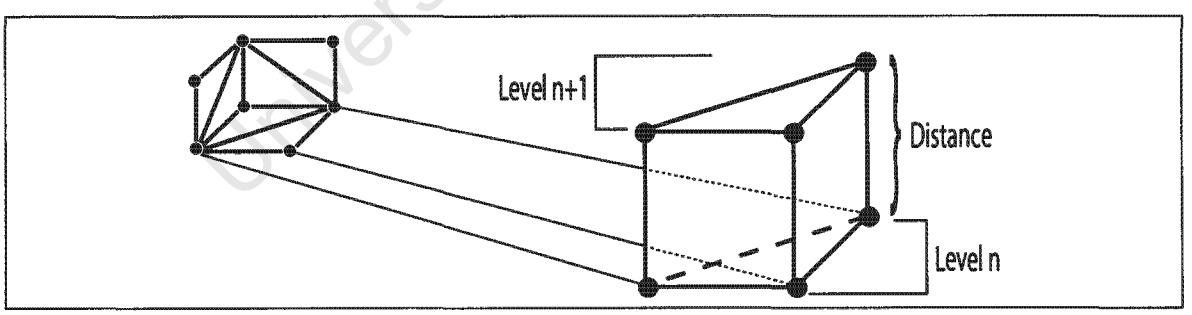


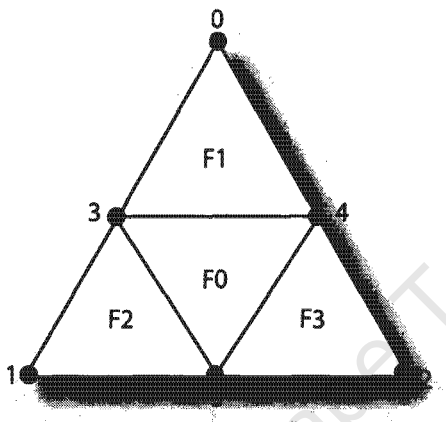
Figure 4.1: **Creating 4D Prisms From 3D Triangles**: Vertices of triangle faces in the 3D model are mapped to their counterpart in a 4D level. 4D vertices are then mapped to their equivalent in the next level thus producing a prism with 4D vertices.

Triangles are a popular choice for the connection information of a 3D mesh for the following reasons: [1]

[1] We assume non-degenerate triangles.

Example 4.1 - Calculating neighbour information

Assume the following 2D model, neighbours for face F_0 are to be calculated:



The following edges are created from the faces and inserted into the edge array (edge $(x, y)_{F_x}$ denotes an edge comprised of vertex indices x and y derived from face F_x):

- F_0: $(3,4)_{F_0}$, $(3,5)_{F_0}$, $(4,5)_{F_0}$
- F_1: $(0,3)_{F_1}$, $(0,4)_{F_1}$, $(3,4)_{F_1}$
- F_2: $(1,3)_{F_2}$, $(1,5)_{F_2}$, $(3,5)_{F_2}$
- F_3: $(2,4)_{F_3}$, $(2,5)_{F_3}$, $(4,5)_{F_3}$

Edges are sorted in the array and are as follows:

$E = \{(0,3)_{F_1}, (0,4)_{F_1}, (1,3)_{F_2}, (1,5)_{F_2}, (2,4)_{F_3}, (2,5)_{F_3}, (3,4)_{F_0}, (3,4)_{F_1}, (3,5)_{F_0}, (3,5)_{F_2}$
$(4,5)_{F_0}, (4,5)_{F_3}\}$

Edges that are identical are as follows:

$\{(3,4)_{F_0}, (3,4)_{F_1}, (3,5)_{F_0}, (3,5)_{F_2}, (4,5)_{F_0}, (4,5)_{F_3}\}$

Therefore Face F_0 will have the following neighbours:

- **Face Neighbour 1**: F_1 with edge (3,4)
- **Face Neighbour 2**: F_2 with edge (3,5)
- **Face Neighbour 3**: F_3 with edge (4,5)

Note: The ordering of neighbours correspond to the ordering of edges.

- **Convexity**: A triangle is the simplest polytope in 2D (it is a 2D simplex) and is thus convex [3].
- **Planarity**: A plane is defined by three non-collinear vertices, therefore a triangle is co-planar to the plane that its vertices define (i.e. it can only exist in one plane).
- **Tessellation**: A surface can be comprised of a collection of triangles which fit together without overlapping or leaving gaps [61].

This dissertation assumes 3D models to have triangular connectivity. Extrusion must produce a 4D object with connectivity information equivalent to its 3D counterpart.

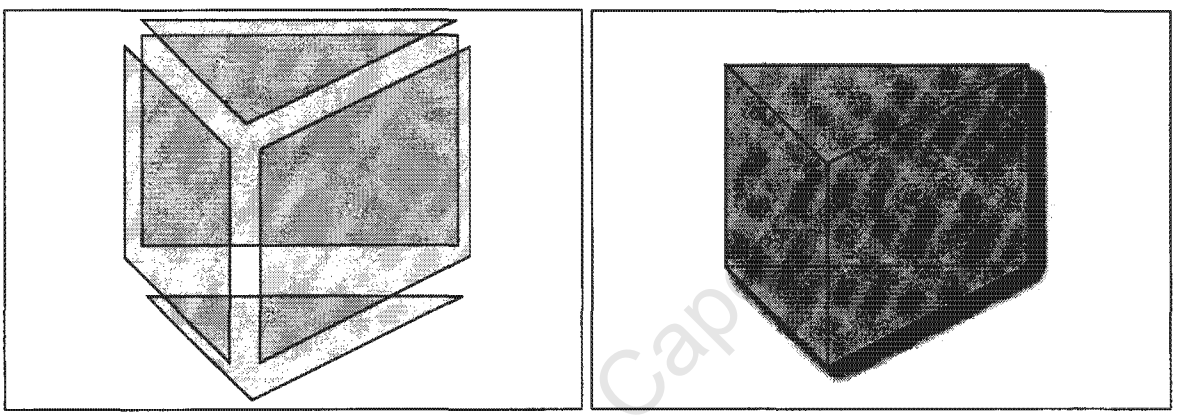


Figure 4.2: **Components Of A Prism** [Left] The 5 components of a prism: Three *quads* and two *triangles* [Right] A prism, with nine edges and six vertices.

4D Connection information is derived from the 3D model's triangle connectivity. Vertices that comprise a triangle are mapped to their 4D equivalent in a particular level. These are then connected to their counterparts in the next level, thus producing a prism with 4D vertices (Figure 4.1). A prism consists of three quads and two triangle faces (Figure 4.2) with nine edges and six vertices. The three edges linking the triangle faces will be referred to as vertical edges in this text.

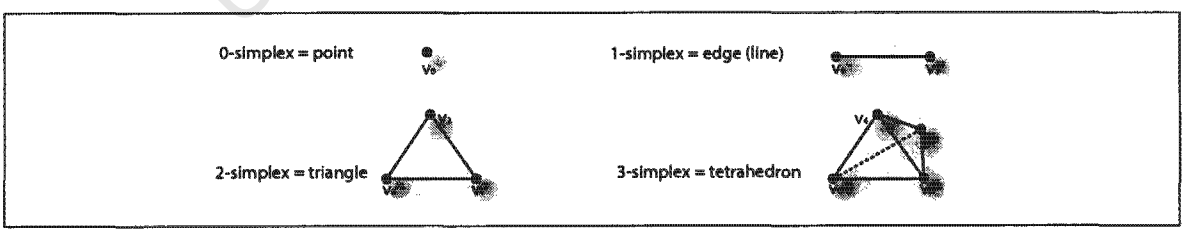


Figure 4.3: **Simplex Sets For The First Four Dimension**: Each n-simplex is the minimal convex set that can exist in n-dimensions.

A prism is not a simplex and therefore not necessarily convex or planar in 4D. Figure 4.4 demonstrates how the quad of a 4D prism can become non-convex under deformation, thus making the

entire prism non-convex. A 3-simplex is the tetrahedron (see Figure 4.3) and is suited to representing a 3D surface in 4D because it has the same properties as a triangle (convexity, planarity and tessellation). This is vital for an unambiguous extraction (see Chapter 6).

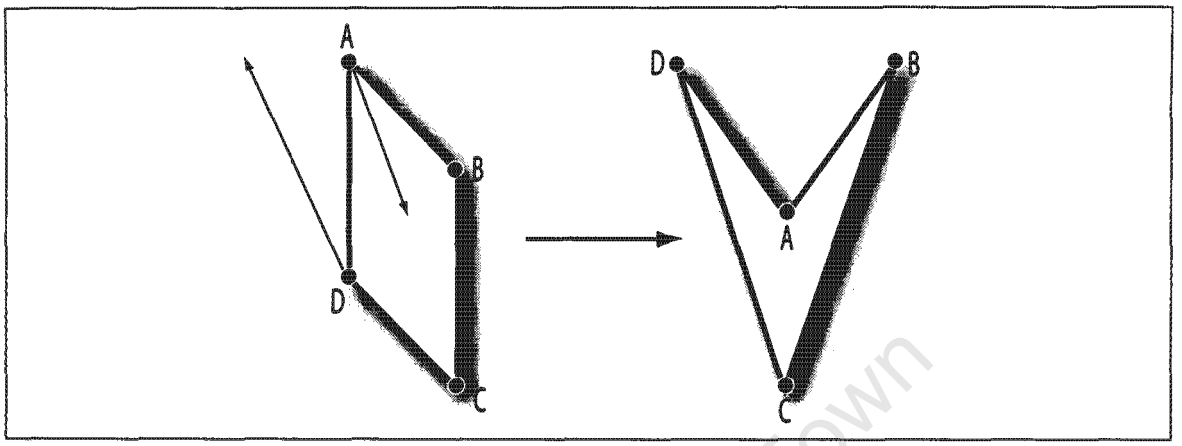


Figure 4.4: **Non-convex Quad After Deformation**: A quad has the potential of becoming non-convex under deformation, thereby making a prism that incorporates it non-convex. [Left] A convex quad before deformation [Right] The resulting deformation which is a non-convex quad.

4.1.4 Subdividing Prisms into Tetrahedra

A simple polyhedron is topologically equivalent[2] to a sphere, thus having polygon faces that are topologically equivalent to a disk [3]. It is well known that simple polyhedra (such as prisms) can be efficiently triangulated[3] without needing to add extra vertices (Steiner vertices) [50]. Tetrahedronalisation[4] is more difficult as Schoeñhardt demonstrated using a prism (Figure 4.6) [19].

Tetrahedronalising a simple polyhedron is accomplished by connecting all vertices via line segments [50]. For a prism, this amounts to splitting the quads diagonally into triangles. The three quads per prism can each be split in two ways (Figure 4.5) resulting in *eight* tetrahedral subdivision schemes.[5] Of these eight schemes, six will subdivide into three tetrahedra (Figure 4.15),while the remaining two will subdivide into four tetrahedra. The two cases that subdivide into four tetrahedra require a Steiner vertex, and are the examples that Schoñhardt demonstrates. These prisms have therefore been given the name Schoñhardt polyhedra and occur when the quads are split with diagonals that do not share a vertex [19] (Figure 4.6). Tetrahedronilsation of a mesh of prisms should produce the following:

- **Simplicial Complex** - The resultant 4D connection information must be a simplicial complex

[2]See Section 3.2.1 for an intuitive explanation into topological equivalence.
[3]Triangulating a polyhedron implies tessellating the polyhedron with triangles.
[4]Tetrahedronalisation implies tessellating a polyhedron with tetrahedra.
[5]There are two methods of splitting a quad into triangles, and there are three quads per prism. Thus there are $2^3 = 8$ ways to tetrahedronalise a prism.

(Section 3.2.2).

- **Minimal** - Tetrahedral subdivision must result in the least number of tetrahedra, and the need to add a Steiner vertex must be avoided.

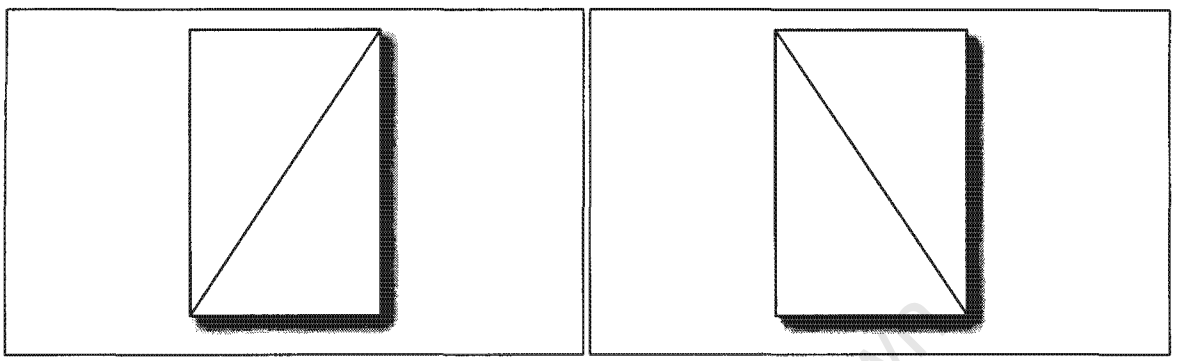


Figure 4.5: **Decomposition of Quadrilaterals into Triangles**: The two ways in which a quadrilateral can be split into a triangle via a diagonal.

The simplicial complex property ensures that the extracted model will be a simplicial complex whilst the minimal property is crucial for ensuring efficiency and memory conservation.

To ensure the minimal property, prism tetrahedronalisation is restricted to three tetrahedra, and throughout this dissertation, prisms will be interchangeably referred to as tetrahedra in *clumps of three*. The number of tetrahedra in a level can thus be determined by multiplying number of faces in the 3D model by three. Tetrahedra emanating from a prism share vertices, edges and faces which are stored adjacently. Determining the index of a tetrahedra makes it trivial to determine the index of its two adjacent neighbours (Section 4.1.3).

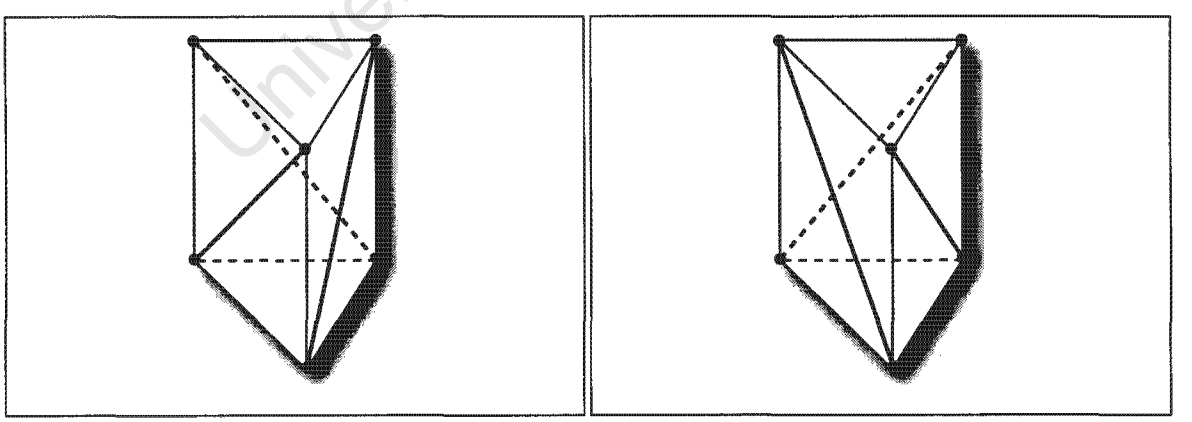


Figure 4.6: **Schoñhardt Polyhedra**: Prisms whose diagonal quad face splits do not share a common vertex.

Minimal Manifold Tetrahedronalisation

The minimal manifold tetrahedronalisation scheme used in this dissertation requires vertices to have a unique identifier with a less-than relationship imposed. [6] Prism quads are split by diagonals that emanate from the *least vertex identifier* [18] in the quad. This scheme ensures a minimal and simplicial complex object:

- **Minimal**: Schoñhardt polyhedra are avoided because two diagonals will always share a vertex. The least index of the entire prism will be the least index of two quads that contain the vertex (Figure 4.7).
- **Simplicial Complex**: Quad faces that are shared by independent prisms must be diagonally split in a consistent manner. Let v_0 be the least index of a quad face Q of prism P_0. Assume Q is also shared by prism P_1, thus v_0 is also shared by prism P_1, and is therefore also the least vertex index of Q in P_1.

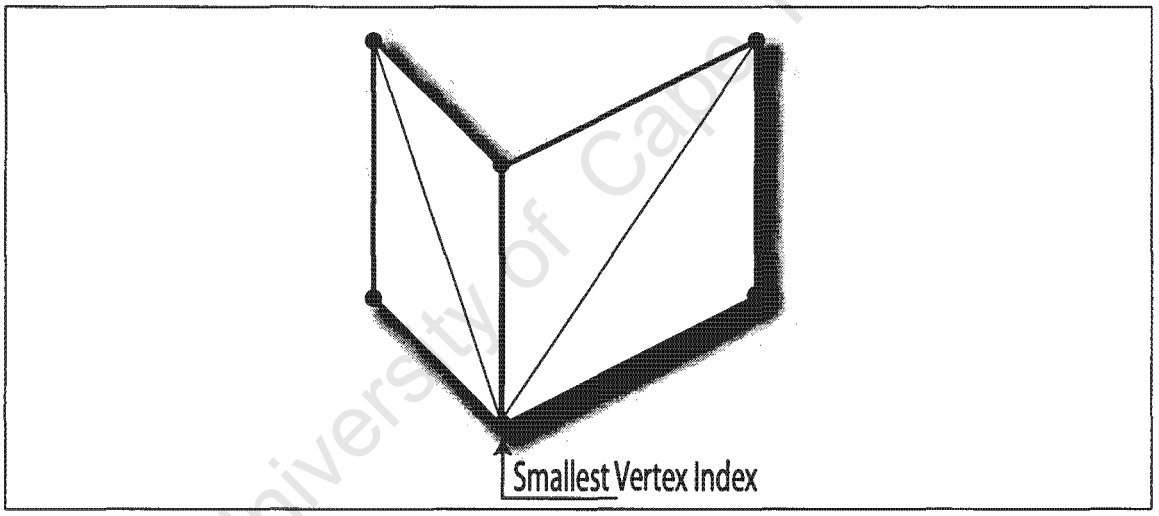


Figure 4.7: **Quad Split From The Least Vertex Identifier**: Prism quads are split using diagonals emanating from the least vertex identifier in the quad. Two quads will share the least vertex identifier of the whole prism, and therefore will contain diagonals emanating from the same identifier

[6] This dissertation uses vertex indices; unsigned integers that have a pre-defined less-than relationship (<).

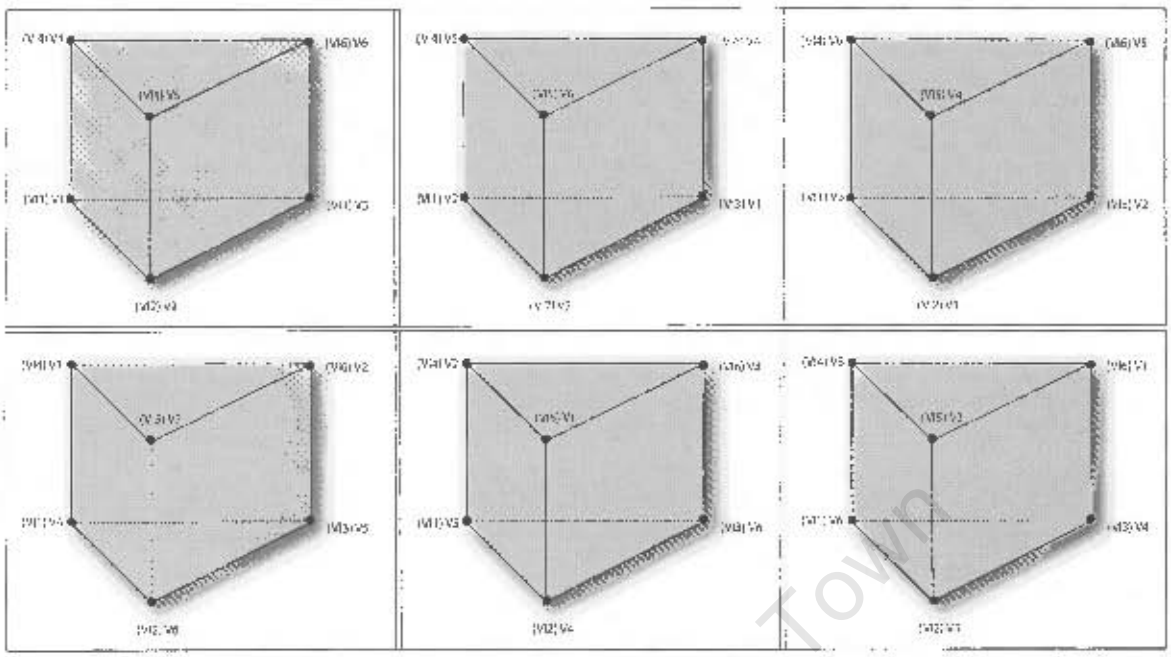


Figure 4.8: **Prism Rotation For Abstraction:** The *six* cases due to treating each one of the six vertices as the least vertex. Each vertex has an abstracted representation in brackets $(V_{I_1} - V_{I_6})$ assigned to vertices $(V_1 - V_6)$ according to table 4.1.

Adopting a level of abstraction makes tetrahedronilastion simpler to describe. The abstraction expresses the prism in a canonical form by using six vertices $V_{I_1}, V_{I_2}, V_{I_3}, V_{I_4}, V_{I_5}, V_{I_6}$ with V_{I_t} always representing the least vertex. The other vertices are given values that reflect the rotation of the prism resulting from assigning V_{I_t} to the least index (Figure 4.8). Table 4.1 is a look-up table used to assign vertex values that reflect the rotation. Expressing quads using four vertices, V_{I_t} will be the least index for the quads $(V_{I_t}, V_{I_2}, V_{I_4}, V_{I_5})$ and $(V_{I_1}, V_{I_3}, V_{I_4}, V_{I_6})$. All that remains is to split the third quad $(V_{I_2}, V_{I_3}, V_{I_5}, V_{I_6})$. The potential diagonals are (V_{I_2}, V_{I_6}) and (V_{I_3}, V_{I_5}), with the diagonal containing the least index being chosen. The following tetrahedra will result:

- $(V_{I_4}, V_{I_6}) < (V_{I_3}, V_{I_5})$:
  - $(V_{I_1}, V_{I_2}, V_{I_3}, V_{I_6})$
  - $(V_{I_1}, V_{I_2}, V_{I_6}, V_{I_5})$
  - $(V_{I_1}, V_{I_5}, V_{I_4}, V_{I_4})$
- $(V_{I_3}, V_{I_5}) < (V_{I_2}, V_{I_6})$:
  - $(V_{I_t}, V_{I_2}, V_{I_3}, V_{I_5})$
  - $(V_{I_t}, V_{I_5}, V_{I_3}, V_{I_6})$
  - $(V_{I_1}, V_{I_5}, V_{I_6}, V_{I_4})$

Smallest Vertex	V_{I_1}	V_{I_2}	V_{I_3}	V_{I_4}	V_{I_5}	V_{I_6}
V_1	V_1	V_2	V_3	V_4	V_5	V_6
V_2	V_2	V_3	V_1	V_5	V_6	V_4
V_3	V_3	V_1	V_2	V_6	V_4	V_5
V_4	V_4	V_6	V_5	V_1	V_3	V_2
V_5	V_5	V_4	V_6	V_2	V_1	V_3
V_6	V_6	V_5	V_4	V_3	V_2	V_1

Table 4.1: **Prism Vertex Abstraction**: Columns represent an abstraction of vertices assuming that V_{I_1} always represents the smallest vertex. This table reflects the values of these abstractions assuming the prism is rotated for V_{I_1} to be assigned the least vertex index (Figure 4.8).

Although there are six tetrahedronalisation schemes (Figure 4.15), there are only two unique schemes (first two rows of Figure 4.15) as shown by the abstraction. The other four are rotations of these two cases. The two unique cases differ in the diagonal chosen for the quad that does not contain the smallest vertex. This can be seen in the abstraction scheme, where tetrahedronalisation relies on what diagonal is chosen for this quad. To distinguish how a prism was tetrahedronalised, it will be categorised as either (V_{I_2}, V_{I_6})**-smaller** or (V_{I_3}, V_{I_5})**-smaller**.

Neighbour Traversal

Memory is conserved by not storing neighbour information for the 4D model. Traversal of 4D neighbours is therefore dependant upon the neighbour information of the 3D model. A tetrahedron face in the 4D model will share edges with nine other faces per level - the three tetrahedra produced from the three prisms that neighbour the prism where the tetrahedron was derived.

The triangle used to construct the prism where the current tetrahedron was derived is determined by taking the modulus of the index of the current tetrahedron by three (integer division - defined for this section by Φ).

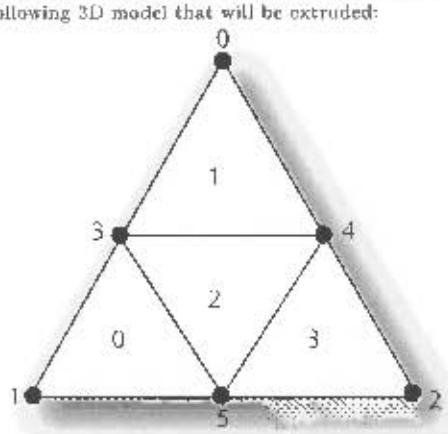
Neighbouring tetrahedra are determined by multiplying neighbour information retrieved from the 3D model by *three* and then adding a constant. This is achieved due to embedding coherence (Section 4.2.2). The calculated reference represents the first tetrahedron derived from the neighbouring prism, with the adjacent two references being the other two neighbouring tetrahedra. Neighbouring 4D prisms therefore differ by a factor of three compared to neighbouring 3D triangles. Assume η is the number of tetrahedra in a level, λ denotes the current level and μ the current tetrahedra index, the neighbouring tetrahedra reference is calculated as follows:

$$N(x) = (\Phi(\mu).getIndexOfNeighbour(x)) \times 3 + \eta \times \lambda \quad x \in \{0, 1, 2\} \tag{4.1}$$

Example 4.2 is an in depth example of 4D neighbour calculation.

Example 4.2 - Determining Tetrahedral Neighbours

Assuming we have the following 3D model that will be extruded:



This example shall determine the tetrahedral neighbours of the tetrahedra that are derived from the prism produced by face 2. Each triangle face will represent one prism per level of the 4D model. Each prism is subdivided into three tetrahedra. Therefore the number of tetrahedra per level (η) is the number of triangle faces multiplied by three:

$\eta = 3 \times 4 = 12$

If the index of the current tetrahedron is 7, then the original triangle face will be $\Phi(7) = 2$. Face 2 has faces 1, 0 and 3 (in that order) as face neighbours. Therefore the nine neighbours per level are as follows:

- $\lambda = 0$:
  - **Neighbour 1** : $N(1) = 1 \times 3 + 0 \times 12 = 3$
    - 3
    - 4
    - 5
  - **Neighbour 2** : $N(0) = 0 \times 3 + 0 \times 12 = 0$
    - 0
    - 1
    - 2
  - **Neighbour 3** : $N(3) = 3 \times 3 + 0 \times 12 = 9$
    - 9
    - 10
    - 11
- $\lambda = x$:
  - **Neighbour 1** : $N(1) = 1 \times 3 + x \times 12 = 3 + 12x$
    - $3 + 12x$
    - $4 + 12x$
    - $5 + 12x$
  - **Neighbour 2** : $N(1) = 0 \times 3 + x \times 12 = 0 + 12x$
    - $0 + 12x$
    - $1 + 12x$
    - $2 + 12x$
  - **Neighbour 3** : $N(3) = 3 \times 3 + x \times 12 = 9 + 12x$
    - $9 + 12x$
    - $10 + 12x$
    - $11 + 12x$

4.1.5 Edge Information

Edge information is stored in the 4D faces using an array structure[7] for efficient random access. Edge information is vital to the extraction process: Edges are used as input to the hyper-plane intersection algorithm (See A.1), the heart of the extraction process. Edge information must comprise the following:

- **Ordering**: Edge ordering is necessary for the production of extracted triangles with correct windings.
- **Consistency**: Two different tetrahedra that share an edge must share a reference to that edge.
- **Efficiency**: The calculation of edge information must be efficient so as to comply with the *interactivity* requirement of virtual sculpting.

Edges are arranged in a specific order to enable consistent retrieval across all tetrahedra. Edge ordering depends upon whether a prism is (V_{I_2}, V_{I_6})**-smaller** or (V_{I_3}, V_{I_5})**-smaller** and is assigned according to Figure 4.10.

Edges assigned to a tetrahedron must be checked against neighbouring tetrahedra already defined. If no neighbouring tetrahedra have a reference to the edge, it is created, with a reference being assigned. Otherwise, the reference is retrieved from the neighbouring tetrahedron.

Retrieval of edge information from neighbours requires that the following be calculated:

- **Neighbour Identification**: Retrieval of all neighbouring tetrahedra using Equation 4.1.
- **Orientation**: The orientation of the neighbouring prism must be determined so as to know which of the three neighbouring tetrahedra to examine.
- **Classification**: The prism's classification of (V_{I_2}, V_{I_6})**-smaller** or (V_{I_3}, V_{I_5})**-smaller** must be determined.

Orientation is ascertained by ordering the vertex indices of the neighbouring prisms and determining if the least vertex index of the neighbouring prism is shared by the current prism (Figure 4.9). It is accomplished by testing the least index from the shared edges of the constituent triangles (Figure 4.2) of the current prism against the ordered vertices of the constituent triangles of the prism neighbours. This is reliant upon 3D neighbour information pertaining to the triangle that produced the current prism. Triangle neighbours one and two (Section 4.1.1) will share edges that are made up from the least vertex index of the triangle. Due to coherence (Section 4.2.2), prism neighbours one and two will also share edges that contain the least vertex index of the prism. Therefore the least vertex of the current prism is tested to determine if it is the least vertex of prism neighbours one and two.

[7] We used the STL vector for our implementation.

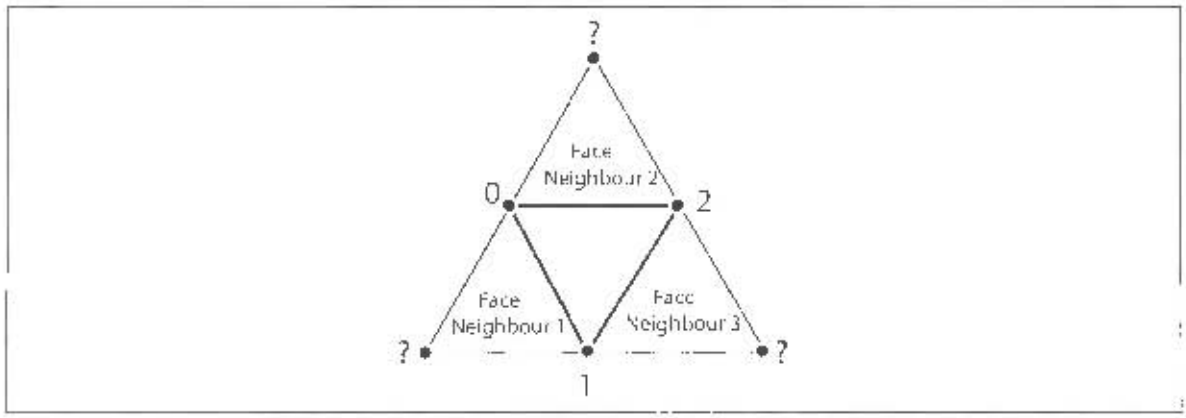


Figure 4.9: **Orientation By Determining Least Vertex Index Of Neighbouring Prism**: Neighbours 1 and 2 will share edges that contain the smallest vertex index - vertex index 0 - of the current prism due to 3D neighbour generation (Section 4.1.1). To determine the orientation of neighbours 1 and 2, the unknown vertices must be tested to determine if they are smaller than vertex index 0. To determine the orientation of neighbour 3, the unknown vertex must be tested against the least vertex index of the shared edge, namely vertex index 1.

Prism neighbour three will share an edge whose least vertex must be determined. It is determined by the prism's classification of (V_{I_2}, V_{I_6})**-smaller** or (V_{I_3}, V_{I_6})**-smaller** since it denotes the smallest vertex of the third edge shared by the third neighbour (Section 4.1.1). The least vertex index is then tested to determine whether it is the least vertex index of the third prism neighbour. There are twelve orientations, illustrated in Figure 4.14.

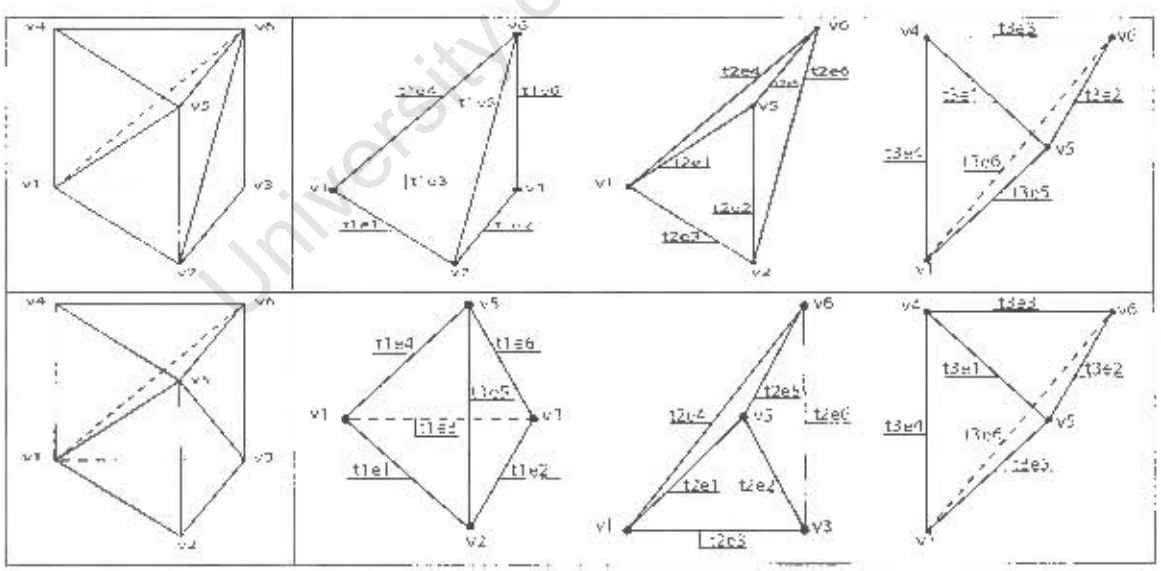


Figure 4.10: **Edge Ordering**: The two unique cases considered for edge ordering. Row 1 is v_2**-smaller** while row 2 is v_3**-smaller**. Notation: txey means tetrahedron x edge index y, where $x \in \{1,2,3\}$ and $y \in \{1,2,3,4,5,6\}$.

The edge creation/assignment algorithm is split into two sections:

1. The creation/assignment of vertical edges of undeformed tetrahedra
2. The creation/assignment of non-vertical edges.

This subdivision makes vertical edge creation/assignment much easier than non-vertical edges because the latter is performed at a different stage; namely the vertex embedding stage, where edge consistency is not important. NOTE: In the following sections, edges will be assigned/created according to Figure 4.10.

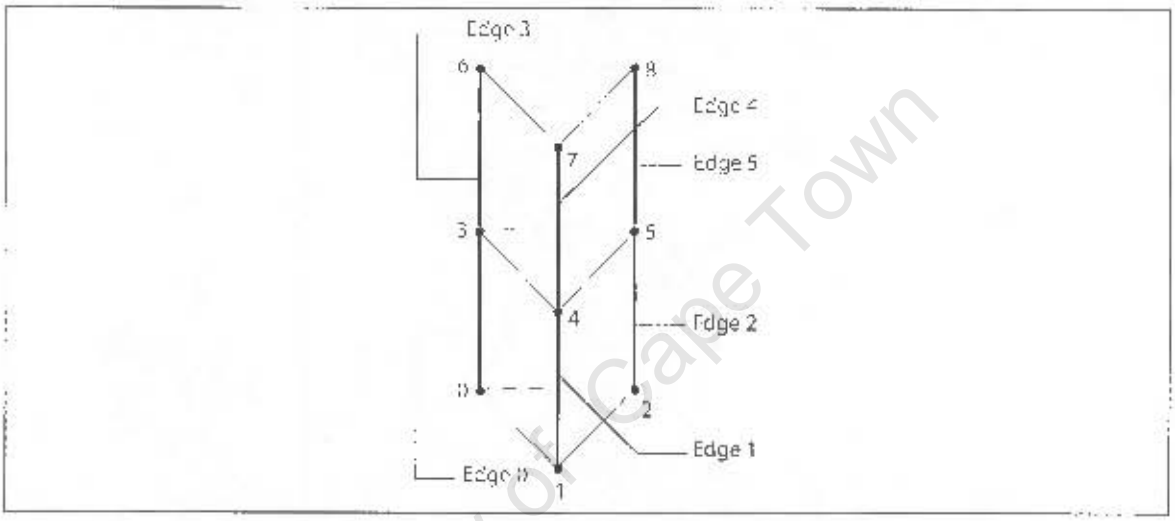


Figure 4.11: **Vertical Edge Reference Assignment:** References for vertical edges are identical to the vertices in the lower layers that comprise them.

Creation/Assignment of Vertical Edges

During the vertex embedding stage, there is no connectivity structure in place therefore assignment does not need to take edge consistency into account. Vertical edges result from vertices in one layer being connected to their counterpart one layer higher (Figure 4.11). These edges are created and assigned references that are identical to the vertices in the base of the prism thus ensuring unique references. Assignment of vertical edges to the prism is trivial since edge references (or edge indices in this case) are determined by examining vertex indices in the lower layer (Figure 4.11).

Vertical edges are assigned as follows (according to Figure 4.11)[8]:

- (V_{l_2}, V_{l_6})**-smaller:** $t_1e_6(v_3, v_6) = v_3$, $t_2e_2(v_2, v_5) = v_2$ and $t_3e_4(v_1, v_4) = v_1$
- (V_{l_3}, V_{l_5})**-smaller:** $t_1e_5(v_2, v_5) = v_2$, $t_2e_6(v_3, v_6) = v_3$ and $t_3e_4(v_1, v_4) = v_1$

[8]**Notation:** t_xe_y means tetrahedron x edge index y, where $x \in \{1, 2, 3\}$ and $y \in \{1, 2, 3, 4, 5, 6\}$.

Creation/Assignment of Non-Vertical Edges

Calculation of non-vertical edges fall into two categories:

- **Edges Already Created:** Edges that have already created by neighbouring tetrahedra, and thus must not be recreated so as to be consistent.
- **Edges Yet To Be Created:** Edges that have not been created by any tetrahedra.

Creating/assigning new edges to tetrahedra is relatively simple. Edges are created and assigned references values corresponding to the next free position (nfp) in the storage structure:

- (V_{I_2}, V_{I_6})**-smaller**
  $t_1e_1(v_1, v_2) = nfp$, $t_1e_2(v_2, v_3) = nfp$, $t_1e_3(v_1, v_3) = nfp$, $t_1e_4(v_1, v_6) = nfp$, $t_1e_5(v_2, v_6) = nfp$
  $t_2e_1(v_1, v_5) = nfp$, $t_2e_3(v_1, v_2) = t_1e_1$, $t_2e_4(v_1, v_6) = t_1e_4$, $t_2e_5(v_5, v_6) = nfp$, $t_2e_6(v_2, v_6) = t_1e_5$
  $t_3e_1(v_5, v_5) = nfp$, $t_3e_2(v_5, v_6) = t_2e_5$, $t_3e_3(v_4, v_6) = nfp$, $t_3e_5(v_1, v_5) = t_2e_1$, $t_3e_6(v_1, v_6) = t_1e_4$
- (V_{I_3}, V_{I_5})**-smaller**
  $t_1e_1(v_1, v_2) = nfp$, $t_1e_2(v_2, v_3) = nfp$, $t_1e_3(v_1, v_3) = nfp$, $t_1e_4(v_1, v_5) = nfp$, $t_1e_6(v_3, v_5) = nfp$
  $t_2e_1(v_1, v_5) = t_1e_4$, $t_2e_2(v_3, v_5) = t_1e_6$, $t_2e_3(v_1, v_3) = t_1e_3$, $t_2e_4(v_1, v_6) = nfp$, $t_2e_5(v_5, v_6) = nfp$
  $t_3e_1(v_5, v_5) = nfp$, $t_3e_2(v_5, v_8) = t_2e_5$, $t_3e_3(v_4, v_6) = nfp$, $t_3e_5(v_1, v_5) = t_2e_5$, $t_3e_6(v_1, v_6) = t_1e_4$

Assigning edge information to tetrahedra where the edges have already been created by a neighbour implies retrieving the edge reference from the neighbour. There are twelve cases to consider, each case dependant upon the prism neighbour's orientation. Note: The following notation is used to to describe each case: $FNX.Y - e_z$ means **F**ace **N**eighbour X tetrahedron Y with edge e_z, where X is one of the three face neighbour prisms, and Y is one of the three tetrahedra that the prism will subdivided into (X,Y $\in \{1,2,3\}$) and edge e_z is one of the size edges in a tetrahedron ($z \in \{1,2,3,4,5,6\}$). Cases one and three of the twelve cases according to Figure 4.14 will be detailed here (see Appendix C for all twelve cases). See Figure 4.12 for a pictorial representation:

- **Case 1**: Neighbour 1 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_1 is the smallest vertex index in Neighbour 1
  $t_1e_1 = FN1.1 - e_3$, $t_2e_1 = FN1.2 - e_4$, $t_3e_1 = FN1.3 - e_3$, $t_2e_3 = t_1e_1$, $t_3e_5 = t_2e_1$
- **Case 3**: Neighbour 1 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_1 is *not* the smallest vertex index in Neighbour 1
  $t_1e_1 = FN1.1 - e_2$, $t_2e_1 = FN1.2 - e_2$, $t_3e_1 = FN1.3 - e_2$, $t_2e_3 = t_1e_1$, $t_3e_5 = t_2e_1$

4.2 Optimisations

Efficiencies in the extrusion process aim to make extrusion as fast as possible. Extrusion is a pre-process and proceeds when the user loads a 3D model. It is not designed to be interactive, with the

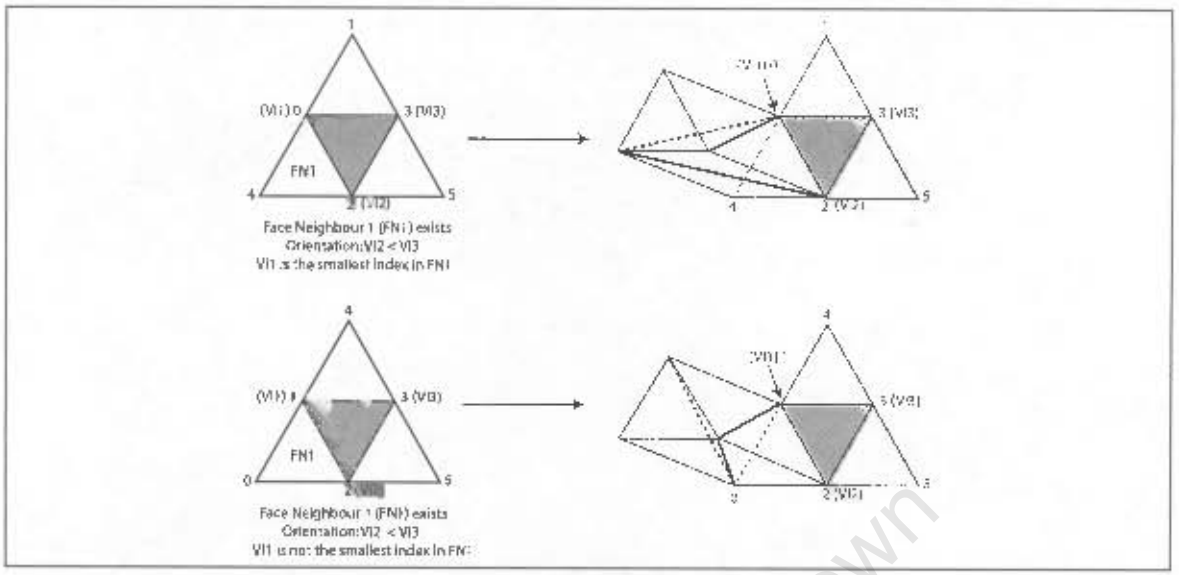


Figure 4.12: **Cases One And Three Of Neighbour Orientation:** (*Note*: Triangle of interest is shaded). Case 1 (Above) will result in a different tetrahedral subdivision compared to case 3 (below). Orientation is thus necessary to choose the correct tetrahedra to obtain edge references from.

small hit in time that it takes deemed acceptable due to a user expecting a time lag immediately after loading the model. Nevertheless, extrusion aims to be as fast as possible to minimise the wait (see Chapter 7 for extrusion results).

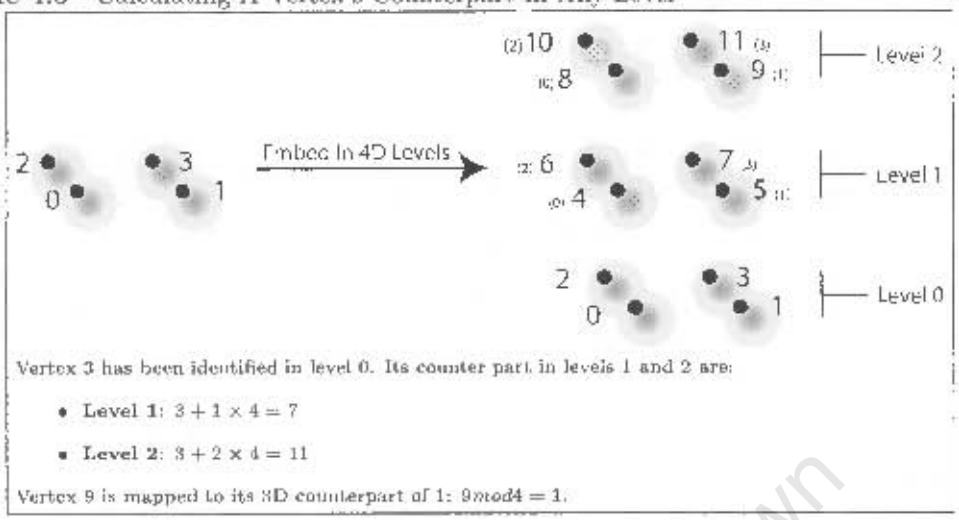
4.2.1 Structural Decisions To Decrease Spatial Redundancy

The 4D model is larger than its 3D counterpart. To save space, information that can be efficiently deduced from the 3D model is discarded from the 4D model. This includes neighbour and normal information, although a full list of edges need be kept for the extrusion process. Edges are more prevalent than vertices and faces, as is evident by Euler's formula. This might affect cache performance, but extraction uses only a small portion of the edges, most of which can be predicted (Section 6.2). Cache is therefore not severely affected. The 4D model is comprised of vertices, edges and tetrahedral faces.

4.2.2 Vertex Embedding Coherence

The 4D model consists of interconnected levels, each level comprising all the vertices of the 3D model with a 4D component added. 3D vertices are embedded in the same order for each level, thereby providing linear coherence. This allows a vertex's counterpart in any level to be calculated by adding a constant comprising the number of vertices in the 3D model multiplied by the level number. Vertices in level 0 have a direct mapping to their 3D counterparts. For example, the third

Example 4.3 - Calculating A Vertex's Counterpart In Any Level



vertex embedded in level 0 will be the third vertex of the 3D model. In the same manner, vertices can be mapped to their 3D counterpart by taking the modulus of its value with the number of vertices in the 3D model (See Example 4.3). Embedding coherence makes the following possible:

- **Vertical Edge Calculation**: Vertical edges can be calculated during the embedding stage without the need for connectivity information. For example, a vertical edge e with vertex index x will also have a vertex index x + number of vertices in 3D model.
- **Tetrahedral Neighbour Calculation**: Determining the 3D triangle that produced the prism where the tetrahedron was derived and determining a neighbour for any levels relies on coherence (Example 4.2).

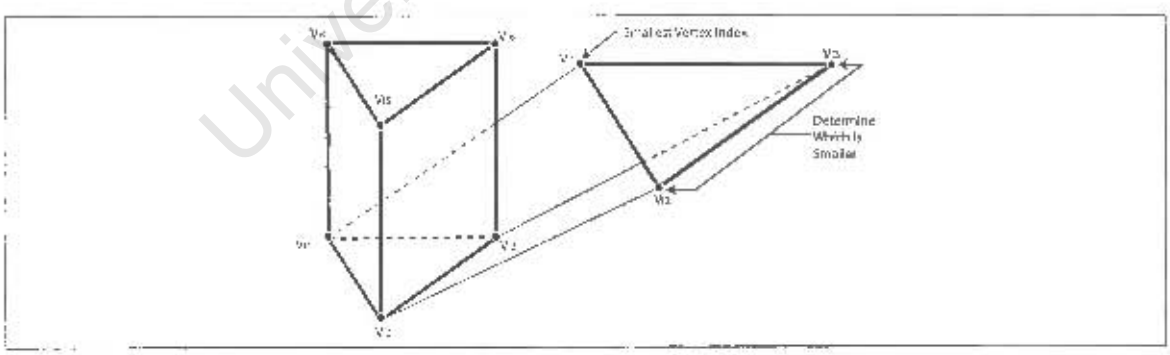


Figure 4.13: **Determining The Least Index Of The Third Quad**: Assuming a prism is rotated such that v_1 is the smallest index; in order to determine which one of the two unique cases for tetrahedral subdivision the prism is, the result of whether v_2 or v_3 is smaller must be calculated

Tetrahedra

Tetrahedronalisation requires that the least vertex of both the prism and third quad be determined. Five comparisons are required to determine V_{I_1}, and another three comparisons to determine if (V_{I_2}, V_{I_5})**-smaller** or (V_{I_3}, V_{I_5})**-smaller**. Due to coherence, vertex indices comprising the triangle in the lower level will be smaller than their higher level counterparts. Therefore vertices comprising the triangle on the higher level can be ignored. Thus three comparisons are required to order the vertices of the lower triangles, with the least vertex being V_{I_1}, and the second least being the least in the third quad. Prisms can therefore be classified as V_{I_2}**-smaller** or V_{I_3}**-smaller**. This has implications for the following:

- **Tetrahedronalisation**: Tetrahedronalisation is more efficient, with only three compares required as opposed to eight.
- **Prism Classification**: Classification is often deduced in conjunction with least vertex determination and can be obtained for free during lower triangle vertex ordering.
- **Extraction**: Extraction optimisations (Section 6.2) require prism classification.

4.2.3 Generalisation of Structure

After building the first level, the 4D object consists of vertices in combination with tetrahedral and edge information for the first level. This information can be extrapolated for all subsequent levels due to coherence. Tetrahedra in subsequent levels can be deduced by adding constants to the vertices that comprise the tetrahedra in the first level. Edges in subsequent levels are calculated in the same manner.

Tetrahedronalisation and edge creation/assignment, the most expensive parts of extrusion, only has to be built for the first level. All other levels are extrapolated from the first level resulting in a considerable improvement in time.

4.3 Summary

An efficient means for creating a 4D model by extruding a 3D model has been presented. Vertices are embedded in 4D, followed by the creation of connection information for each level. Connection information is derived from the 3D model, with triangles connected to their counterparts in subsequent levels to form prisms. Prisms are not simplex, and therefore have the potential to become non-convex under deformation. This has negative implications for the extraction process, therefore prisms are subdivided into tetrahedra, a simplex. Edges are then assigned in a consistent ordered manner to tetrahedra. Optimisations are developed from the coherence of vertex embedding.

The extruded model is now ready to be both deformed and extracted.

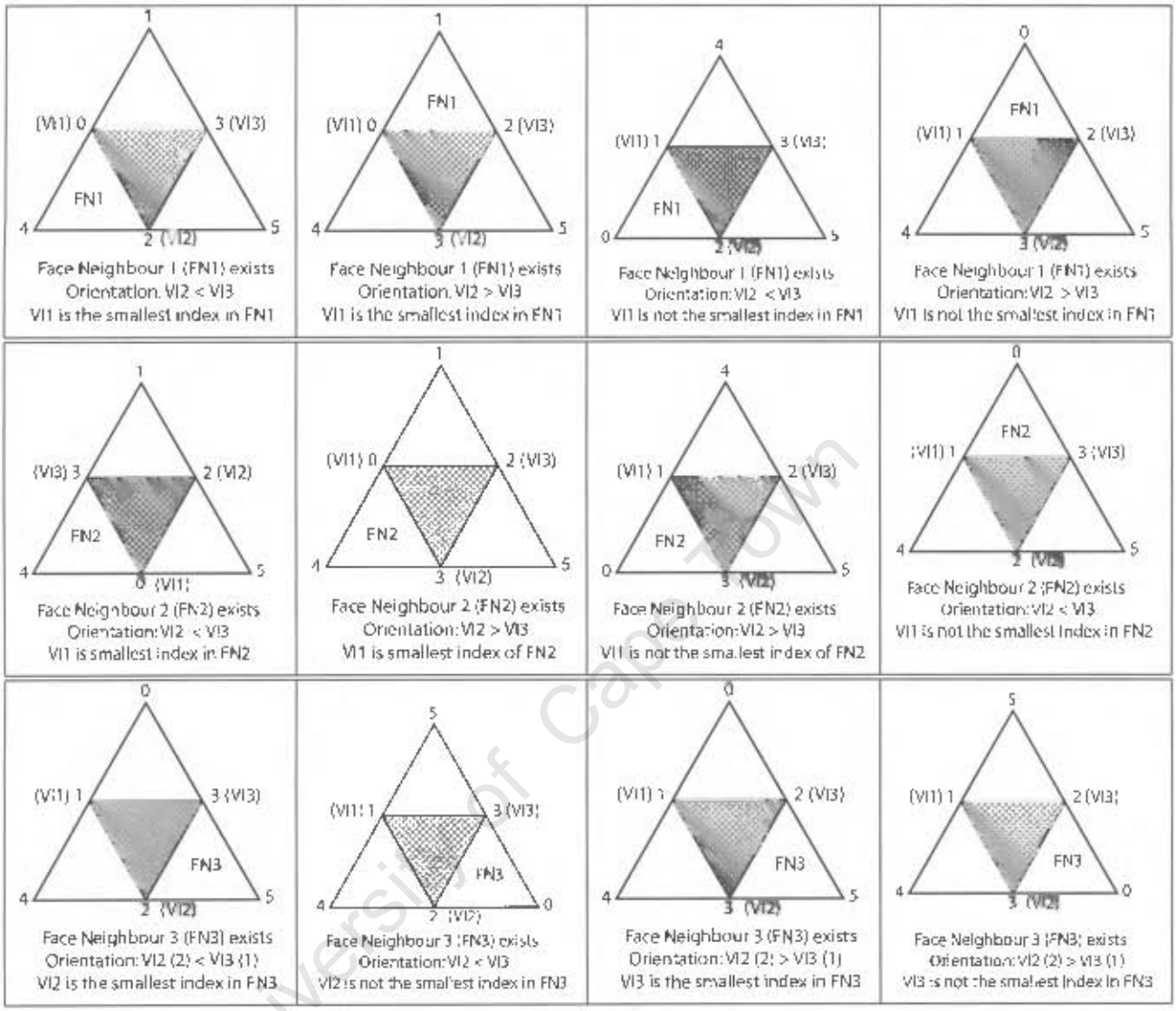


Figure 4.14: **Face Neighbour Cases For Edge Detection** The *twelve* cases that need be considered for detecting neighbouring edges from neighbouring tetrahedra when building tetrahedron edges (Note: Current face of interest is shaded in *grey*)

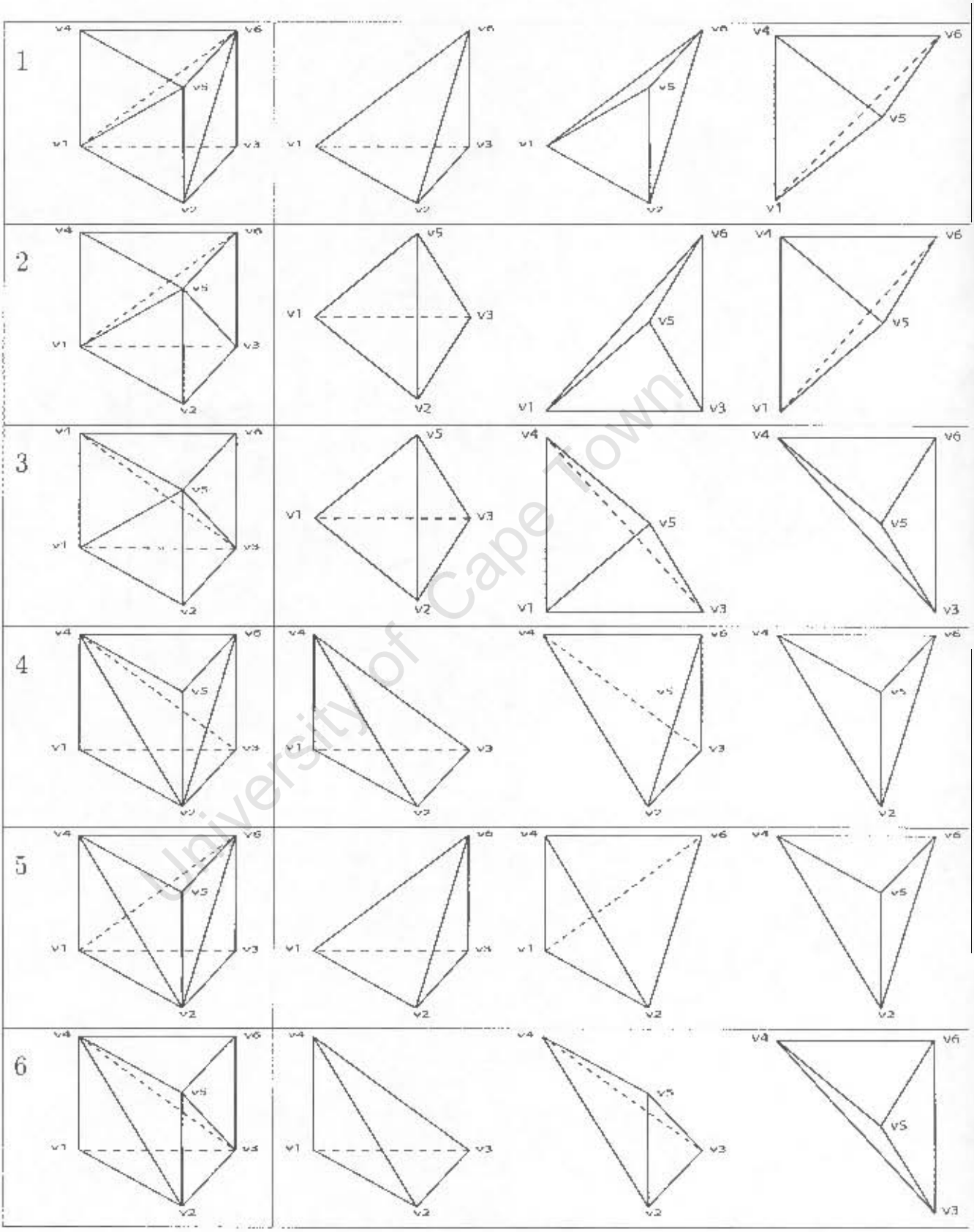


Figure 4.15: **Tetrahedral Decomposition of Prisms**: The *six* possible ways to decompose a prism into three tetrahedra. Note: Only case case 1 and 2 are unique, all other cases are rotations of these two.

Chapter 5

Deformation

The deformation process is detailed in this chapter. A form of spatial deformation known as Direct Manipulation of Free-Form Deformation (DMFFD) is the chosen means with which to warp objects. DMFFD is a spatial deformation, it therefore results in object primitives (vertices) attaining new spatial positions. This dissertation uses triangle meshes as the computerised model for deformation.

The chapter is partitioned as follows:

- **Section 5.1 - The Deformation Process** - A high level overview of the deformation process is presented. The process can be subdivided into two subprocesses, Free-Form Deformation (FFD) and Direct Manipulation of Free-Form Deformation (DMFFD):
  - **Section 5.1.1 - Free-Form Deformation:** The algorithm and mathematics behind FFD is presented.
  - **Section 5.1.2 - Direct Manipulation Of Free-Form Deformation:** A means by which DMFFD works in conjunction with FFD is explained along with the mathematics behind DMFFD.
- **Section 5.2 - Adapting Deformation To Four Dimensions:** 3D deformation must be adapted to 4D in order to warp extruded objects.
  - **Section 5.2.1 - Adapting FFD To 4D:** The mathematical changes needed to adapt FFD from 3D to 4D are explained.
  - **Section 5.2.2 - Adapting DMFFD To 4D:** Differences between the mathematics of 4D DMFFD as compared to 3D are detailed.
- **Section 5.3 - Optimisations:** Optimisations required to make deformation interactive are explained:
  - **Section 5.3.1 - Previous Optimisations:** Optimisations that 4D deformation uses as a foundation to build upon.

- **Section 5.3.2 - 4D Deformation Optimisations:** 4D deformation optimisations that help in the overall topology alteration process.

- **Section 5.4 - Summary**

5.1 The Deformation Process

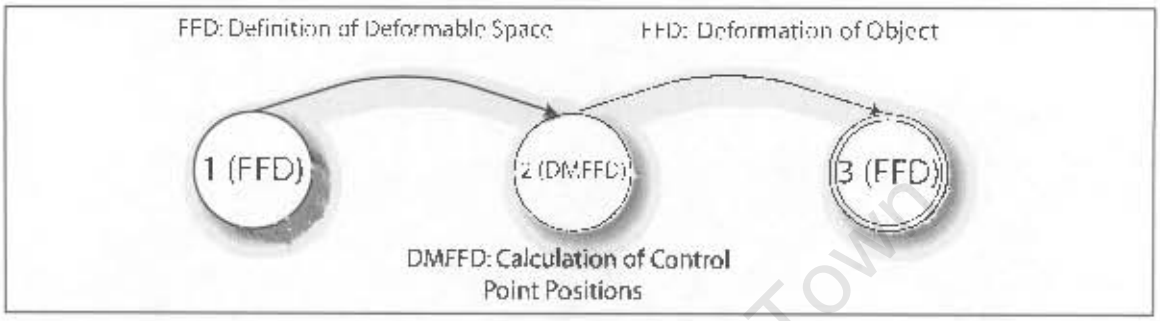


Figure 5.1: **The Deformation Process State Machine:** [1] Setting up the deformable space by the FFD process [2] The adjusting of the control points by the DMFFD process [3] The deformation process as performed by FFD with new control point positions.

Deformation is implemented using a point based scheme [26] known as Direct Manipulation of Free-Form Deformation (DMMFD) [35] (Section 3.1.3). DMFFD is an efficient and simple interface to a hyper-patch warping technique called Free-Form Deformation [55] (Section 3.1.2). DMFFD can be decoupled from FFD allowing each process to be explained separately.

The FFD process is composed of the following sub-processes:

- **Cuboid Coordinate System:** A coordinate system is defined on a cuboid volumetric space.
- **Hyper-patch Definition:** The cuboid volume is subdivided by imposing a lattice of control points. The collective volume described by the control points in conjunction with the spline basis is known as the *hyper-patch*.
- **Vertex Embedding:** Object vertices are converted from $\mathbb{R}^3$ into the cuboid coordinate system.
- **Control Point Translation:** The lattice control points are spatially translated, thereby warping the hyper-patch.
- **Deformation:** The warping effect of the hyper-patch is carried through to the embedded object.

The DMFFD process consists of the following sub-processes:

- **Feature Specification**: A point within the hyper-patch is given a post-deformation spatial position.
- **Control Point Reconfiguration**: The FFD lattice control points are reconfigured to define a deformation that reflects the features.

Even though DMFFD and FFD are decoupled, they cannot be performed in isolation. Deformation is performed in the following order (Figure 5.1):

1. **Definition of Deformable Space**: The first two sub-processes of FFD.
2. **Calculation of Control Point Positions**: The DMFFD process.
3. **Deformation of Object**: The last three sub-processes of FFD.

5.1.1 Free-Form Deformation

Cuboid Coordinate System

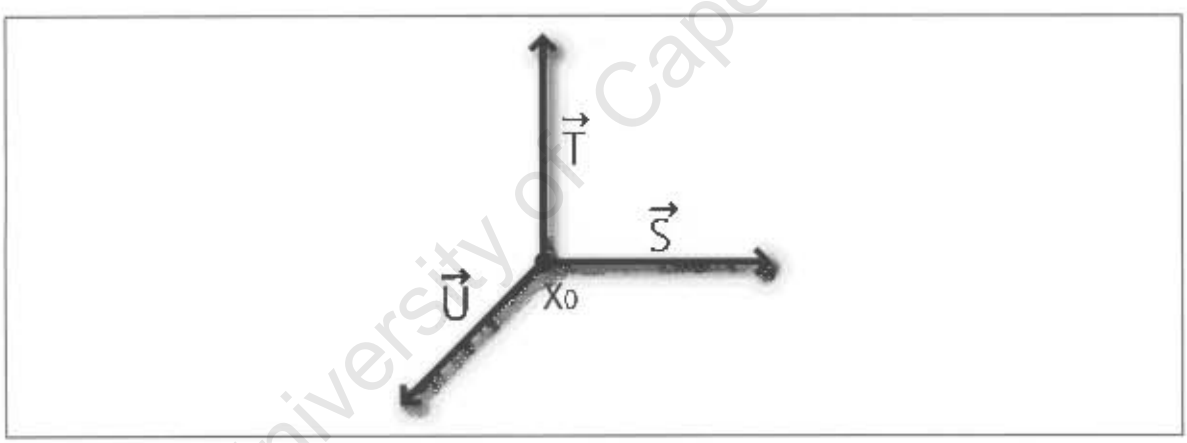


Figure 5.2: **Cuboid Coordinate System**: The origin (X_0) and three orthogonal vectors $\vec{S}, \vec{T}, \vec{U}$ comprise the cuboid coordinate system.

FFD is a spatial deformation and therefore objects intended for deformation must be embedded in its deformable space. This requires the deformable space to have a coordinate system imposed upon it. It consists of a point describing the origin and three vectors (see figure 5.1.1):

- X_0: The origin point, representing the origin of the deformable space with respect to the the world space origin.
- $\vec{S}$: Vector that spans the length of the deformable space.
- $\vec{T}$: Vector that spans the height of the deformable space.

- $\vec{U}$: Vector that spans the breadth of the deformable space.

The coordinate system as described by Sederberg and Parry in [55] is more general than the one specified above since it is able to describe any parallelpiped volume. Restricting the vectors $\vec{S}, \vec{T}$ and $\vec{U}$ to be orthogonal results in a cuboid deformable space that makes embedding more efficient (Section 5.1.1).

Hyper-patch Definition

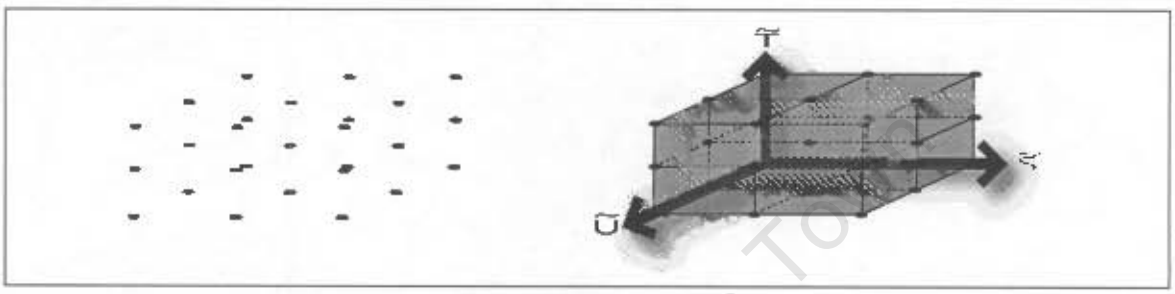


Figure 5.3: **Control Point Lattice**: [Left] A lattice of control points [Right] The space described by lattice control points superimposed on the cuboid coordinate system.

A lattice of control points is imposed on the cuboid volume (Figure 5.3). The control points are analogous to spline control points and the lattice should be thought of as a generalisation of a control polygon for spline interpolation/approximation in 3D. Each individual deformable volume that a portion of control points, in conjunction with the spline basis describes, is known as a cell. The collective volume that the cells describes is called the hyper-patch. The hyper-patch is not the convex hull of the lattice points since the nature of the b-spline basis function implies that some of the lattice points will not be involved with the hyper-patch (Figure 5.4). The hyper-patch is the FFD instance of the class of deformable spaces belonging to spatial deformations.

Assuming there are l, m and n control points for each of the $\vec{S}, \vec{T}, \vec{U}$ dimensions, there will then be $l \times m \times n$ control points in total with each control point P_{ijk} being the i'th j'th k'th control point along the $\vec{S}, \vec{T}, \vec{U}$ dimensions respectively.

Vertex Embedding

A point $\mathcal{X} = (x, y, z)$ in world space is expressed in hyper-patch deformable space by adding a parametric combination of $\vec{S}, \vec{T}, \vec{U}$ to X_0 [55]:

$$\mathcal{X} = X_0 + s\vec{S} + t\vec{T} + u\vec{U} : s, t, u \in \mathbb{R} \tag{5.1}$$

The (s, t, u) values for $\mathcal{X}$ can be calculated by using the following equations [55]:

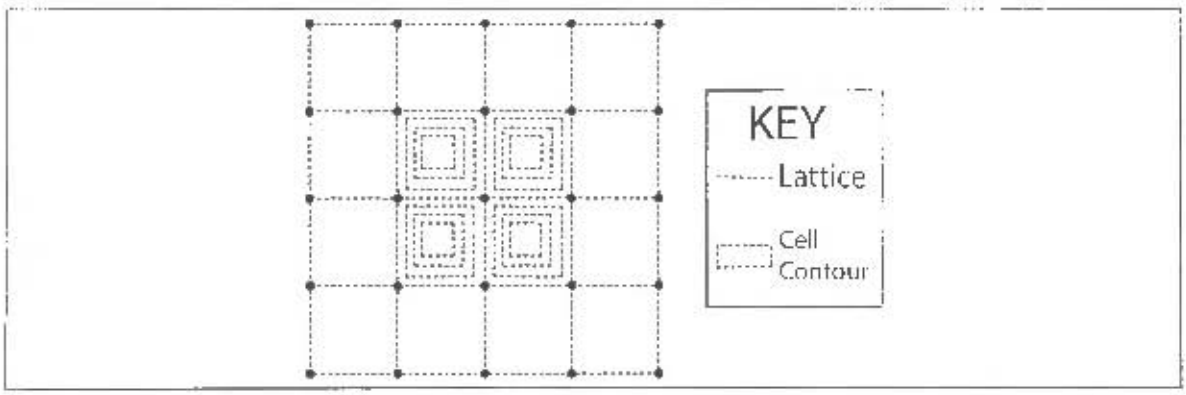


Figure 5.4: **Lattice Hyper-Patch Relationship In Two Dimensions**: The hyper-patch is the deformable volume described by the lattice. Not every lattice point will be part of the hyper-patch. The nature of the basis function used for deformation in conjunction with the lattice points will define the hyper-patch.

$$\begin{aligned} s &= \frac{(\vec{T} \times \vec{U}) \cdot (\mathcal{X} - X_0)}{\vec{T} \times \vec{U} \cdot \vec{S}} \\ t &= \frac{(\vec{S} \times \vec{U}) \cdot (\mathcal{X} - X_0)}{\vec{S} \times \vec{U} \cdot \vec{T}} \\ u &= \frac{(\vec{S} \times \vec{T}) \cdot (\mathcal{X} - X_0)}{\vec{S} \times \vec{T} \cdot \vec{U}} \end{aligned} \tag{5.2}$$

The above equations assume a general parallelepiped deformable volume, but they can be simplified by restricting the volume to a cuboid shape (Section 5.1.1) (See Appendix B for a proof of these equations):

$$\begin{aligned} s &= \frac{\vec{S} \cdot (\mathcal{X} - X_0)}{\vec{S} \cdot \vec{S}} \\ t &= \frac{\vec{T} \cdot (\mathcal{X} - X_0)}{\vec{T} \cdot \vec{T}} \\ u &= \frac{\vec{U} \cdot (\mathcal{X} - X_0)}{\vec{U} \cdot \vec{U}} \end{aligned} \tag{5.3}$$

The embedding equation $\mathcal{E}$ that embeds $\mathcal{X}$ to cuboid coordinates $\mathcal{U}$ can be expressed as follows:

$$\mathcal{E}(\mathcal{X}) = (s, t, u) = \mathcal{U} \tag{5.4}$$

If $\mathcal{X}$ occurs in the deformable space, then $s, t, u \in [0, 1]$. Points outside the deformable space (e.g. points with $s, t, u \in (-\infty, 0) \cup (1, \infty)$) will cause extrapolation errors and are ignored. Equation 5.4 with its composite equations (Equations 5.3) form the FFD embedding equation, and are the FFD instances of Equation 3.1. Hence an embedded object has had the world coordinates of its vertices converted to the local deformable coordinate system.

Control Point Translation

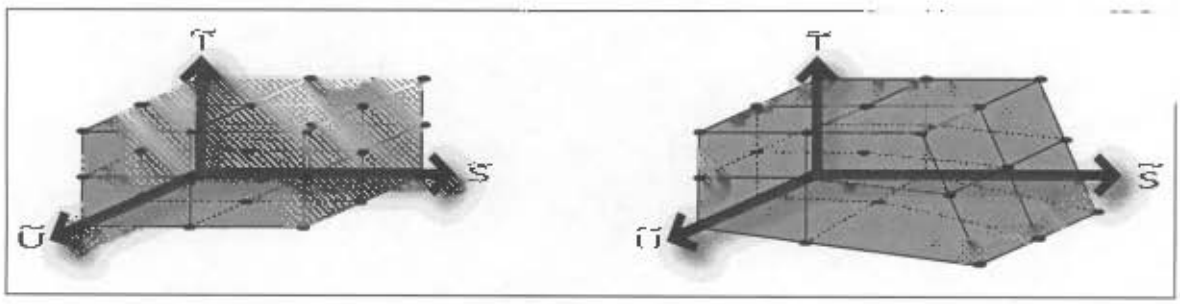


Figure 5.5: **Control Point Translation**: Control points are given new spatial positions (translated) thereby distorting the deformable space which they described. [Left] Lattice before control points are translated [Right] Warped lattice resulting from translating the right most control points.

Control points P_{ijk} are given new spatial positions resulting in a redefinition (warping) of the hyper-patch (Figure 5.5). The warping effect on the hyper-patch is carried through to the embedded object. Deformation is thus achieved by altering the control points of the lattice while the mesh is left alone. It is for this reason that FFD is considered to indirectly deform the object.

Deformation

Deformation warps a point $\mathcal{X}$ in world space to $\bar{X}$, its warped equivalent. It is realised by evaluating embedded object points within the hyper-patch as a weighted sum of the control points, with each control point weighted by a polynomial B-Spline basis function $\mathcal{N}$. Let a vector, known as a knot vector be defined as a non-decreasing sequence $\{t_0, t_1, \ldots, t_m\}$ with $t_i \in [0,1]$, then a B-Spline basis is defined as follows:

$$\mathcal{N}_0^i(t) = \begin{cases} 1 & \text{if } t_i < t < t_{i+1} \text{ and } t_i < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{N}_p^i(t) = \frac{t - t_i}{t_{i+p} - t_i}\mathcal{N}_{p-1}^i(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}}\mathcal{N}_{p-1}^{i+1}(t)$$

This dissertation makes use of a cubic B-Spline basis, which has the following attractive properties:

- **Flexible**: The cubic polynomial is efficient due to its low degree while it models the increased derivative continuity and flexibility of higher degrees relatively well [26].
- **Local Control**: The B-Spline basis allows for local control, meaning that individual cells and not the entire hyper-patch will be affected by a warping.

The FFD instance of the deformation function (Equation 3.2) becomes:

$$\mathcal{H}(\mathcal{E}(\mathcal{X})) = \mathcal{H}(s,t,u) = \sum_{i=0}^{a+l-1}\sum_{j=0}^{b+m-1}\sum_{k=0}^{c+n-1} \mathcal{N}_i^l(s).\mathcal{N}_j^m(t).\mathcal{N}_k^n(u).P_{ijk} = \bar{X} \qquad (5.5)$$

See Figure 3.1 for a pictorial example of deformation.

5.1.2 Direct Manipulation of Free-Form Deformation

DMFFD belongs to the class of point-based deformations since deformations are specified using points within the deformable space. It therefore directly warps an object, providing an easy interface to FFD. While there are many configurations the control points can attain to satisfy the specified deformations, DMFFD provides a solution that changes the configuration in a least squared sense (Figure 5.6).

Feature Specification

A feature is a point constraint within the hyper-patch that specifies its position after deformation [46]. The DMFFD model allows the user to specify multiple features per deformation. A feature consists of two variables:

- $\mathcal{C}$: The origin of the feature in world space. $\mathcal{C}$'s embedded location must occur in the interior of the hyper-patch.
- $\Delta\mathcal{C}$: The displacement vector representing the intended position of the feature after deformation.

Features can be considered as input to the DMFFD process, whose output is a control point reconfiguration.

Control Point Reconfiguration

The post deformation position of each feature must be $\mathcal{C}_x + \Delta\mathcal{C}_x$. Expressing this using Equation 5.5:

$$\forall x, \mathcal{H}(\mathcal{E}(\mathcal{C}_x)) = \sum_{i=0}^{a+l-1} \sum_{j=0}^{b+m-1} \sum_{k=0}^{c+n-1} \mathcal{N}_i^a([\mathcal{E}(\mathcal{C}_x)]_s).\mathcal{N}_j^m([\mathcal{E}(\mathcal{C}_x)]_t).\mathcal{N}_k^n([\mathcal{E}(\mathcal{C}_x)]_u).P_{ijk} = \mathcal{C}_x + \Delta\mathcal{C}_x \tag{5.6}$$

In the above equation, there is only one set of unknowns: The spatial positions of P_{ijk}. The aim is to find spatial positions for P_{ijk} such that $\forall x, \mathcal{H}(\mathcal{E}(\mathcal{C}_x)) = \mathcal{C}_x + \Delta\mathcal{C}_x$. P_{ijk} can be expressed as:

- $\mathcal{P}_{ijk}$: The initial spatial position in the lattice before deformation.
- $\Delta\mathcal{P}_{ijk}$: The displacement vector describing its new position after deformation.

Therefore $P_{ijk} = (\mathcal{P}_{ijk} + \Delta\mathcal{P}_{ijk})$. Equation 5.6 can now be rewritten as follows:

$$\forall x, \mathcal{C}_x + \Delta\mathcal{C}_x = \sum_{i=0}^{a+l-1} \sum_{j=0}^{b+m-1} \sum_{k=0}^{c+n-1} \mathcal{N}_i^a([\mathcal{E}(\mathcal{C}_x)]_s).\mathcal{N}_j^m([\mathcal{E}(\mathcal{C}_x)]_t).\mathcal{N}_k^n([\mathcal{E}(\mathcal{C}_x)]_u).(\mathcal{P}_{ijk} + \Delta\mathcal{P}_{ijk}) \tag{5.7}$$

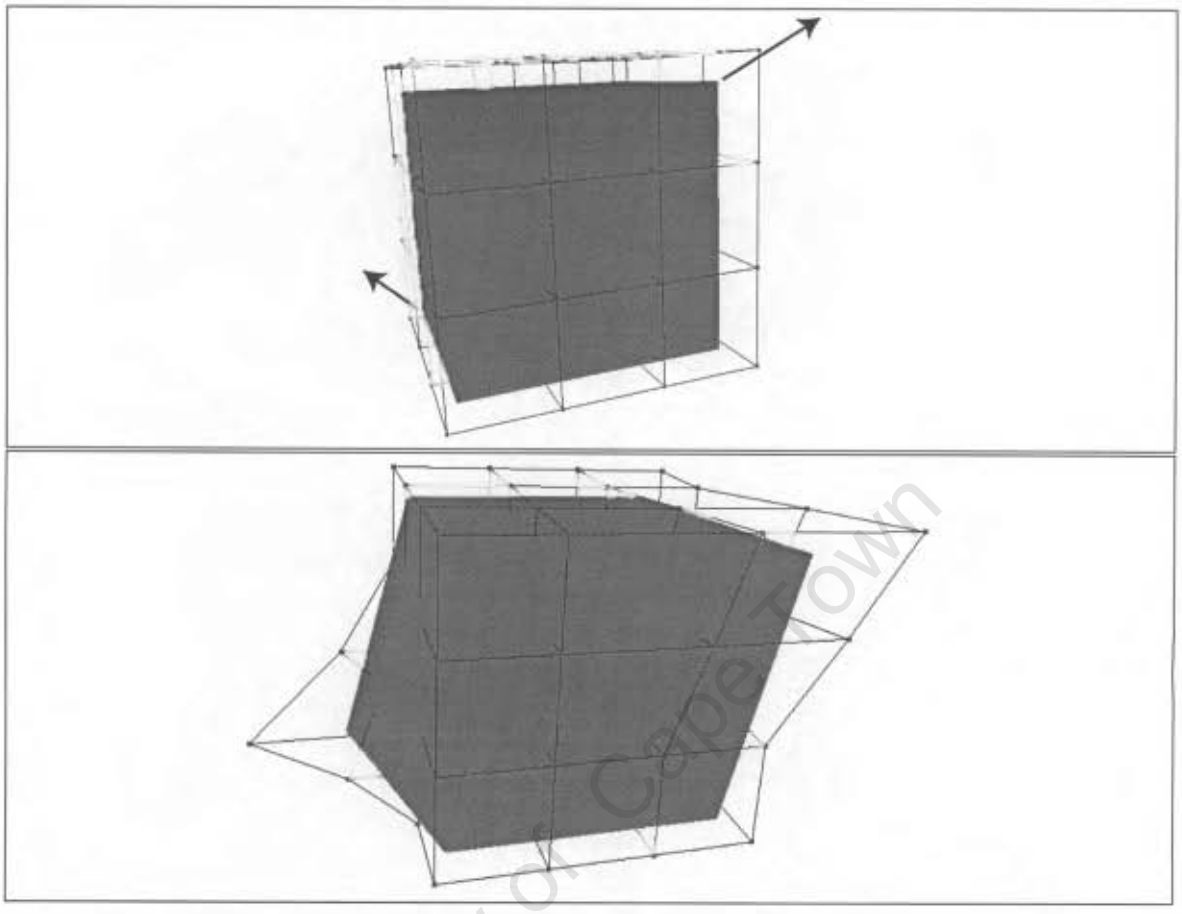


Figure 5.6: **DMFFD**: [Above] An object is embedded in a point lattice with features specifying post deformation positions [Below] The deformed object resulting from the warped lattice. The warped lattice's control points are repositioned in a least squared sense.

A series of linear equations is imposed on the above equation. That is, each variable in the above equation is placed in matrix form. It now becomes:

$$\mathbf{C} + \mathbf{\Delta C} = \mathbf{B}(\mathcal{P} + \mathbf{\Delta}\mathcal{P})$$

$$\Rightarrow \mathbf{\Delta C} = \mathbf{B}\mathbf{\Delta}\mathcal{P} \tag{5.8}$$

- Each C_x and ΔC_x is placed in a $r \times 3$ matrix $\mathbf{C}$ and $\mathbf{\Delta C}$ respectively. The rows represent the number of features and their displacements vectors.
- Each control point $\mathcal{P}_{ijk}$ and their associated control vector $\Delta\mathcal{P}_{ijk}$ is placed in matrix $\mathcal{P}$ and $\mathbf{\Delta}\mathcal{P}$ respectively. The matrices have dimensions $s \times 3$, with rows representing control points and its associated displacement vectors.
- The B-Spline basis functions $\mathcal{N}$ are expressed in matrix $\mathbf{B}$. The dimensions of this matrix are $r \times s$, with weight entries in column j influencing row j of $\mathcal{P}$ and $\mathbf{\Delta}\mathcal{P}$.

Solving equation 5.8 will yield the desired values for $\Delta\mathcal{P}$. This is achieved by calculating an inverse for B ($\mathbf{B^{-1}}$):

$$\mathbf{B^{-1}}.\mathbf{B}\Delta\mathcal{P} = \mathrm{B}^{-1}.\Delta\mathbf{C}$$

$$\Rightarrow \Delta\mathcal{P} = \mathbf{B^{-1}}.\Delta\mathbf{C}$$

It is assumed that **B** is non-singular, since a singular matrix will be the result of user errors (e.g. specifying two features for one point) that can be checked for. There are three possible states that **B** can attain:

1. **B is exact:** If **B** is square ($r = s$) and non-singular, an exact inverse $\mathbf{B^{-1}}$ can be found.
2. **B is under-determined:** If ($r < s$), **B** is under-determined. This occurs when there are more unknowns in $\Delta\mathcal{P}$ than features. There are infinite configurations pertaining to the new spatial positions of P_{ijk} that will satisfy the features. An example of when this would happen is when users specify less features than there are numbers of control points.
3. **B is over-determined:** If ($r > s$), **B** is overdetermined. This will occur when there are more features than unknowns in $\Delta\mathcal{P}$. For example, if a user a user tries to create an undulating surface that has more undulations than control points [35]. There is no spatial configuration for the control points that will satisfy all the features.

Since it is not possible to produce an exact inverse for the last two cases, a formulation known as the pseudo-inverse (denoted by $\mathbf{B^+}$) is used:

$$\Delta P \approx \mathbf{B^+}\Delta C \tag{5.9}$$

The pseudo-inverse of **B** is a unique matrix calculated by [35]:

- ***Under-Determined***: $\mathbf{B^+} = B^T(BB^T)^{-1}$.
- *Over-Determined*: $\mathrm{B}^+ = B(B^TB)^{-1}$.

It has many desirable properties:

- If **B** is a square non-singular matrix, then the pseudo-inverse formulation will produce the exact inverse.
- If **B** is under-determined, a configuration that makes $\| \Delta\mathcal{P} \|$ as close to zero as possible is calculated.
- If **B** is overdetermined, a configuration that minimises the error (i.e. $\| B\Delta\mathcal{P} - \Delta C \|$ is as close to zero as possible) is calculated.

5.2 Adapting Deformation To Four Dimensions

Extruded objects (Chapter 4) exist in four dimensions, and deformation must therefore be adapted to four dimensions. An overview of the 4D deformation process is presented with emphasis on amendments to 3D deformation. Adapting deformation to four dimensions implies adapting both FFD and DMFFD to 4D. Adapting FFD to 4D consists of:

- **Hyper-Cuboid Coordinate System:** A coordinate system that is generalised to 4D must be produced.
- **Hyper-patch Definition**: The hyper-cuboid system is subdivided by imposing a lattice of 4D control points to define a deformable volume in 4D.
- **4D Vertex Embedding**: Vertices are embedded from global 4D space to the hyper-cuboid space.
- **4D Control Point Translation:** Analogous to the 3D case, 4D control points are spatially altered resulting in the hyper-patch becoming warped.
- **4D Deformation:** Deformation of the hyper-patch is carried through to the embedded object.

While adapting DMFFD to 4D includes:

- **4D Feature Specification:** Features are expressed in 4D, therefore both $\mathcal{C}$ and $\Delta\mathcal{C}$ are expressed in 4D.
- **4D Control Point Reconfiguration**: 4D Control points are reconfigured to affect a deformation that reflects the features.

5.2.1 Adapting FFD To 4D

Hyper-Cuboid Coordinate System

A coordinate system describing a hyper-cuboid space defines the 4D embeddable space. The coordinate system is almost identical to the 3D system, with the following alterations:

- **Four Dimensions**: The origin X_0 and vectors $\vec{S}, \vec{T}, \vec{U}$ are expressed in 4D ($X_0, \vec{S}, \vec{T}, \vec{U} \in \mathbb{R}^4$) by appending a fourth component.
- **Extra Vector:** A vector $\vec{V}$ that is orthogonal to $\vec{S}, \vec{T}, \vec{U}$ is created to describe the fourth dimension.

The origin scalar X_0 measures displacement from the 4D origin $(0,0,0,0)$. The $\vec{S}, \vec{T}$ and $\vec{U}$ vectors measure the length, breadth and height in 4D. Length, breadth and height are in the 3D hyperspace, therefore the appended 4D component is 0. The vector $\vec{V}$ being orthogonal to $\vec{S}, \vec{T}$ and $\vec{U}$ describes 4D component (Example 5.1).

Example 5.1 - Hyper-Cuboid Coordinate System

Assume the 3D case. The cuboid coordinate system could be defined by the following:

- $X_0 = (1, 1, 1)$
- $\vec{S} = (1, 0, 0)$
- $\vec{T} = (0, 1, 0)$
- $\vec{U} = (0, 0, 1)$

The analogous hyper-cuboid coordinate system could be defined as follows:

- $X_0 = (1, 1, 1, 1)$
- $\vec{S} = (1, 0, 0, 0)$
- $\vec{T} = (0, 1, 0, 0)$
- $\vec{U} = (0, 0, 1, 0)$
- $\vec{V} = (0, 0, 0, 5)$

Note: The $\vec{V}$ represents an extruded object with extrusion height 5.

Hyper-Patch Definition

4D Control points are imposed on the hyper-cuboid coordinate system to define the deformable hyper-patch. The 4D lattice differs from 3D in the following ways:

- **Four Dimensions**: Control points are expressed in 4D and have an extra index to represent the extra hyper-cuboid vector ($\vec{V}$). Therefore, if there are l, m, n and o control points for each of the $\vec{S}, \vec{T}, \vec{U}, \vec{V}$ dimensions, there will be $l \times m \times n \times o$ control points in total with P_{hijk} representing the h'th i'th j'th and k'th control point.
- **Extra Dimension**: Control points are set along four dimensions, and therefore are placed along an extra vector, $\vec{V}$.

4D Vertex Embedding

Vertices that are embedded in the hyper-cuboid coordinate system are described by local coordinates in 4D. An equation that provides for the $\vec{V}$ dimension is appended to the embedding equations (Equations 5.3):

$$v = \frac{\vec{V} \cdot (\mathcal{X} - X_0)}{\vec{V} \cdot \vec{V}} \tag{5.10}$$

The embedding equation is therefore:

$$\mathcal{E}(\mathcal{X}) = (s, t, u, v) = \mathcal{U} \tag{5.11}$$

4D Deformation

The deformation equation takes into account the fourth dimension:

$$\mathcal{H}(\mathcal{E}(\mathcal{X})) = \mathcal{H}(s,t,u,v) = \sum_{h=0}^{a+l-1} \sum_{i=0}^{b+m-1} \sum_{j=0}^{c+n-1} \sum_{k=0}^{d+o-1} \mathcal{N}_h^a(s).\mathcal{N}_i^m(t).\mathcal{N}_j^n(u).\mathcal{N}_k^o(v).P_{hijk} = \hat{X} \tag{5.12}$$

5.2.2 Adapting DMFFD To 4D

4D Control Point Reconfiguration

Equation 5.7 is expressed in 4D:

$$\forall x, \mathcal{H}(\mathcal{E}(\mathcal{C}_x)) = \sum_{h=0}^{a+l-1} \sum_{i=0}^{b+m-1} \sum_{j=0}^{c+n-1} \sum_{k=0}^{d+o-1} \mathcal{N}_h^a([\mathcal{E}(\mathcal{C}_x)]_s).\mathcal{N}_i^m([\mathcal{E}(\mathcal{C}_x)]_t).\mathcal{N}_j^n([\mathcal{E}(\mathcal{C}_x)]_u).\mathcal{N}_k^o([\mathcal{E}(\mathcal{C}_x)]_v).(P_{hijk} + \Delta P_{hijk}) = \mathcal{C}_x + \Delta\mathcal{C}_x$$

As in Equation 5.7, the only set of unknowns is $(P_{hijk} + \Delta P_{hijk})$. Converting the above equation to a series of linear equations is identical to Equation 5.8 with the following differences:

- $\mathbf{C}$ and $\mathbf{\Delta C}$ have dimensions $r \times 4$.
- $\mathcal{P}$ and $\mathbf{\Delta}\mathcal{P}$ have dimensions $s \times 4$.

A 4D pseudo-inverse B^+ is created, with the same properties as its 3D counterpart, and is used to calculate new spatial positions of the 4D control points.

5.3 Optimisations

Optimisations ensure that the deformation process does not hinder any of the properties needed for virtual sculpting (Section 3.1), specifically the interactivity property. Efficiencies for 3D virtual sculpting, developed by Gain [25, 26], are used as a foundation for deformation improvements. Additional optimisations pertaining to 4D deformation are also developed.

5.3.1 Previous Efficiencies

Efficiencies that were developed from Gain include [25, 26]:

- **Implicit Definition Of Control Points**: Hyper-patch control points are not explicitly stored in memory. Instead, they are calculated by exploiting the coherence of a point lattice. All that is stored is the origin of the lattice, and the space between control points.

- **Efficient Pseudo-Inverse Calculation**: The pseudo-inverse (B^+) is the single most computationally intensive task within DMFFD. Making it efficient is essential for the interactivity requirement of virtual sculpting. A combination of normal equations and exploitation of the sparseness of B produces an efficient calculation of the pseudo-inverse (See pp 51-54 of [25] and pp 52-55 of [26] for a more detailed explanation).
- **Efficient Basis Polynomial Calculation**: The costly recursive nature of B-Spline polynomial evaluation (known as Cox-de Boor recursion) can be deprecated by a more efficient direct polynomial evaluation provided that knots are uniformly spaced in the knot vector ([20], pp 153-156).
- **Only Deforming Points Within Warped Cells**: The nature of the B-Spline polynomial basis implies that a local deformation is performed during warping. Therefore, not all object points will be deformed, and only those points in cells undergoing warping need be calculated using Equation 5.5.

Although these optimisations were implemented for 3D, they are trivially extended to 4D.

5.3.2 4D Deformation Efficiencies

Tetrahedron Tagging

The extraction process will benefit immensely if the deformation process returns a list of deformed tetrahedra. Tetrahedra that are not deformed do not have to be extracted as extraction will result in triangles identical to the ones in the 3D model used to build it. Deformed tetrahedra require extraction since the three tetrahedra that comprise the prism may not be be aligned to the hyper-plane after deformation, and therefore produces a point cloud that will need an unambiguous triangulation. The extraction process can therefore substitute the original model's triangle information for the undeformed tetrahedra and append to it information extracted from the deformed tetrahedra to make an efficient and accurate extraction.

The deformation process tags all tetrahedra that are deformed, and returns a vector containing references to the tagged tetrahedra.

Embedding Coherence

It is assumed that the original 3D object (see Chapter 4) is already embedded in a deformable space. This information can be used to produce embedding information for the 4D object. The extruded object consists of levels of 3D objects, each level with a different 4D offset. The only difference between a point and its counterpart in the next level is the 4D offset. By analogy, the only difference between an embedded point and its counterpart in other embedded levels is the 4D offset. An extruded object is therefore embedded by providing 4D offset information to the 3D embedded object. Calculation of 4D offsets are efficient due to the coherence of the point lattice.

Even though obtaining embedded information is efficient, it takes up a large amount of space to embed all the 4D model's points. 4D Points are therefore embedded only when required, which is just before deformation. Points in levels that are not deformed are therefore not embedded.

5.4 Summary

The deformation process, as it pertains to virtual sculpting and topology alteration has been explained. Deformation consists of FFD and DMFFD, with DMFFD providing an intuitive interface to FFD. The mathematics behind both FFD and DMFFD have been detailed for the 3D and 4D cases. Optimisations are presented that make deformation interactive and help the topology alteration process, specifically the extraction stage.

The 4D model, after it has been deformed is now ready to be used in the extraction process.

Chapter 6

Extraction

Extraction is the final stage of the topology alteration process. It aims to extract a topologically altered 3D model from the extruded 4D model. This is achieved by intersecting a 3D hyper-plane with the 4D model resulting in a 3D point set that is then triangulated. Optimisations are introduced to make the process interactive so as not to violate the required properties of virtual sculpting.

This chapter is partitioned as follows:

- **Section 6.1 - The Extraction Process:** The extraction process is explained, with emphasis on it being a hierarchical process; the hierarchy consisting of the *model*, *tetrahedra* and *edges*. It is divided into two parts:
  - **Section 6.1.1 - Point Set:** A point set is created from hyper-plane 4D object intersection.
  - **Section 6.1.2 - Triangulation:** The point set is triangulated to form a 3D model.
- **Section 6.2 - Efficiencies:** The speed of the extraction process is improved making it interactive. Efficiencies pertain to each level of the extraction hierarchy:
  - **Section 6.2.1 - Efficiencies At The Edge Level**
  - **Section 6.2.2 - Efficiencies At The Tetrahedral Level**
  - **Section 6.2.4 - Efficiencies At The Model Level**
- **Section 6.3 - Summary**

6.1 The Extraction Process

Obtaining an extracted 3D model is accomplished in two steps: A 3D point set is *constructed* by intersecting a hyper-plane with the 4D model after which, the point set is *triangulated*.

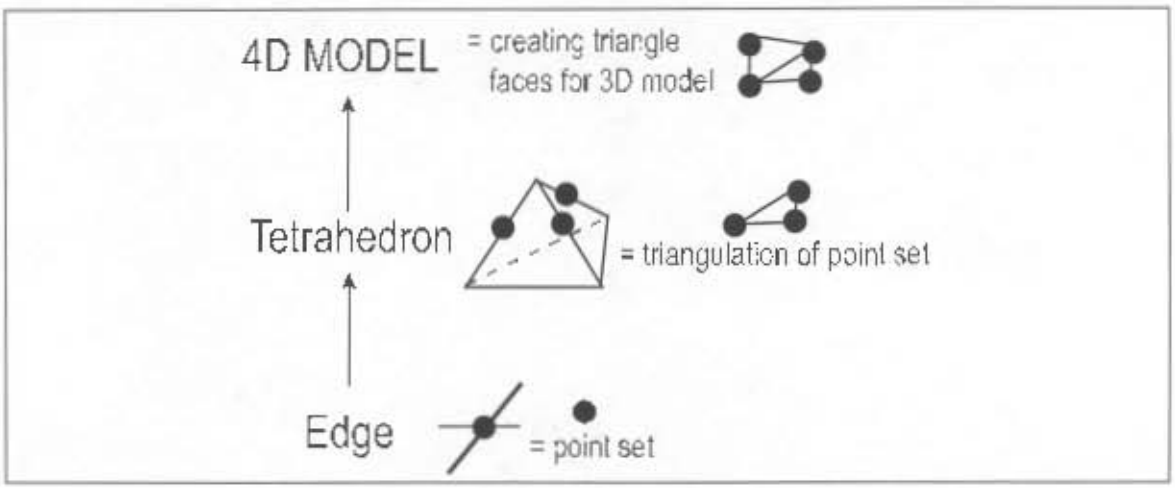


Figure 6.1: **The Extraction Hierarchy**: At the bottom of the hierarchy are the *edges*, from which points are extracted. In the middle is the tetrahedra where the point set is triangulated. At the top is the model, where triangle faces for the 3D model are created.

Extraction is a three levelled hierarchical process (Figure 6.1):

1. **The Model**: At the top of the hierarchy is the 4D model. The 4D model consists of vertices, edges and tetrahedra (Section 4.2.1): all the elements required by extraction.
2. **Tetrahedra**: Next in the hierarchy are the individual faces that make up the model the tetrahedra. Tetrahedra are composed of four vertices and six edges.
3. **Edges**: At the bottom of the hierarchy are the edges. Edges are passed to the hyper-plane edge intersection algorithm where a point is produced if an intersection occurs.

The extraction algorithm attempts to produce points for each of the six edges of a tetrahedron, for each tetrahedron in the 4D model. This is only achievable if the hyper-plane intersects the edges of the tetrahedron. The production of the point set occurs at the edge level, triangulation at the tetrahedra level and creation of the 3D model at the model level (Figure 6.1).

Properties Of The Model

The 4D model is the result of extruding (Chapter 4) a 3D model. Like the 3D model, the 4D model is a manifold simplicial complex (Section 3.2.3 and Section 3.2.2) consisting of vertices, edges and tetrahedra. To conserve space, no normal vector information is stored in the 4D model.

Properties Of The Tetrahedron Used In Extraction

A non-degenerate tetrahedron is a fully connected polyhedron consisting of four unique vertices. (Figure 6.2). The following properties make a non-degenerate tetrahedron pertinent to hyper-plane intersection:

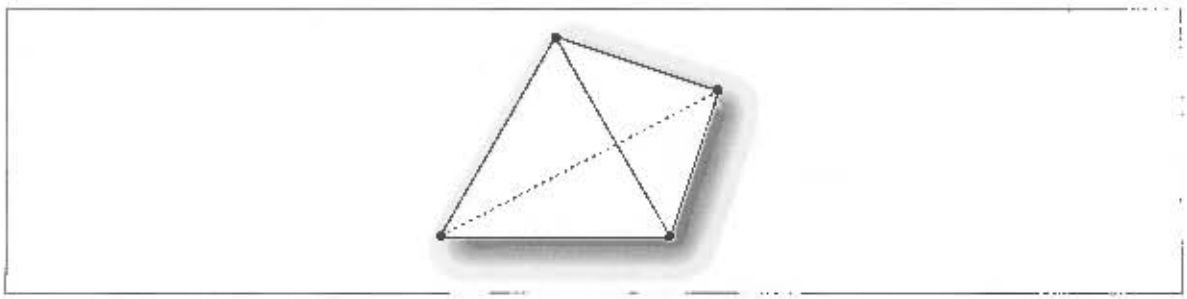


Figure 6.2: **A Non-degenerate Tetrahedron**: A fully connected polyhedron consisting of four unique vertices.

- **Planarity**: The four unique vertices that constitute a non-degenerate tetrahedron defines a 3D hyper-plane. Non-degenerate tetrahedra are therefore embedded in exactly one hyper-plane in 4D.
- **Convexity**: Non-degenerate tetrahedra are convex. When intersected with a hyper-plane, a point set that lies on a convex hull is produced which can be unambiguously triangulated. Hyper-plane intersection with a non-convex polyhedra produces a point set whose connection is ambiguous (Figure 6.3).

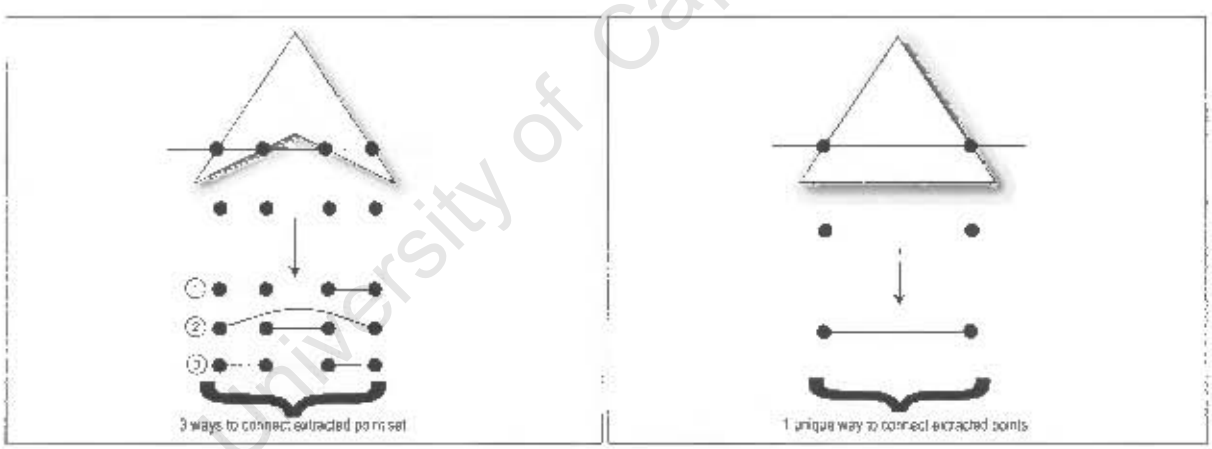


Figure 6.3: **Non-Convex/Convex Polyhedra Plane Intersection**: [Left] A non-convex polyhedra will produce a point set that is ambiguous. There are multiple ways to connect the extracted points, only one is a correct connection (connection 1) [Right] A convex polyhedra intersected with a hyper-plane will produce a point set that can only be connected in one unambiguously correct way

The planarity and convexity conditions of a tetrahedron do not hold for a prism. Under deformation, the quads that constitute the prism may become non-convex and non-planar. *Planarity* and *convexity* ensure that an extracted point set with the following characteristics is produced:

- **Unambiguous Connection**: An unambiguous connection scheme can be determined due the convexity of a tetrahedron (Figure 6.3 demonstrates that a plane intersection with a non-convex shape will produce connection ambiguities).

- **Convexity:** The extracted point set will lie on a convex hull. The resulting connection will produce a convex polygon, amenable to a generalised triangulation scheme.
- **Non-linearity:** Points will not be linear combinations of each other. Therefore a non-degenerate convex point set whose connection is a straight line will not be produced.

Properties Of Edges Used In Extraction

An edge in Euclidean space is defined as an unordered pair of end points which specify a line segment that connects them [33]. Any point on the line can be expressed as a linear combination of the two end points. If an edge is defined by the end points X_0 and X_1, a point x on the edge can be expressed by the linear parametric equation $x = (1-t)X_0 + tX_1$, where $t \in [0,1]$. The value of t determines the position on the line where x is situated. For example, when $t = 0$, $x = X_0$, and when $t = 1$, $x = X_1$ (Figure 6.4). This implies that if $t < 0$ or $t > 1$, x is not on the edge (line segment).

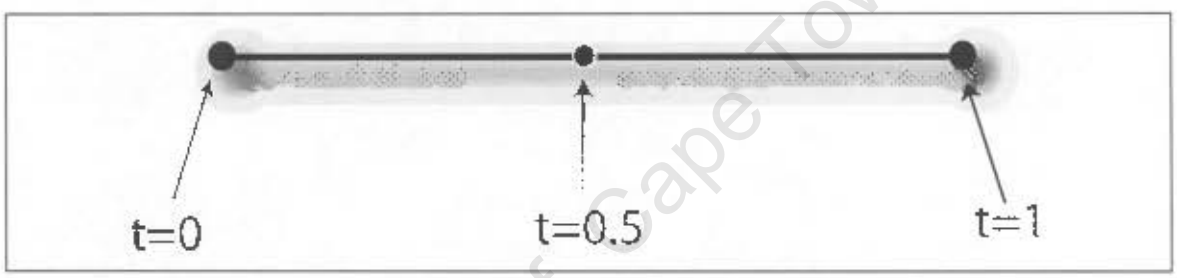


Figure 6.4: **Points On An Edge:** The position of a point on an edge is determined by the value of t pertaining to the linear parametric equation $x = (1-t)X_0 + tX_1$, where $t \in [0,1]$ When $t = 0$, the point is the beginning end point, when $t = 0.5$, it is in the middle and when $t = 1$, it is the last end point.

6.1.1 Constructing the Point Set

Extracted vertices are obtained by intersecting the edges of the 4D model with a 3D hyperplane. The algorithm that intersects an edge with a hyper-plane (Section A.1) produces a value for t that denotes the position of the intersection (Figure 6.5) when used with the linear parametric equation of an edge (Section 6.1). If $t \notin [0,1]$, the hyper-plane does not intersect the edge and can therefore be ignored. Section A.1 presents a detailed explanation of the hyper-plane edge intersection algorithm.

The point produced by applying t to the linear parametric equation is in 4D. To acquire a 3D vertex, the 4D vertex must be projected into 3D. An orthogonal projection in w is used because it is efficient and the 4D component is not required. The projection is implemented by removing the 4D component from the point. This implies that points only differing by their 4D components will be mapped to the same 3D coordinate. But all undeformed points intersected by a hyper-plane orthogonal to the fourth dimension will have the same 4D components, and differ everywhere else. This may not be the case however after the 4D object has been deformed: A deformation could

result in the 4D object self intersecting thereby producing 4D points that may project onto each other. For topology alteration, this is avoided by strictly controlling the deformations so that this situation does not arise. Deformations that produce self intersections are normally the result of a user specification.

Edges shared between neighbouring tetrahedra must not make separate extractions for each tetrahedron as this will result in multiple vertices with the same coordinates. An incorrect geometry for the 3D model will then be produced, leading to rendering problems such as tears. This is avoided by storing a reference to the extracted point in the edge and a flag that indicates an extracted point has been produced. Neighbouring tetrahedra share references to shared edges and can thus easily determine if an edge has already undergone extraction and obtain its extracted point. Note: In general, edges are avoided as places to store information due to the fact that there are so many edges in a model; this is the only instance where information is stored in an edge.

6.1.2 Triangulation

The aim of triangulation is to produce a *geometry* consisting of triangles that tessellate the point set resulting from the edge hyper-plane intersection. It is required for rendering properties such as lighting and back face calculation. A non-degenerate tetrahedron intersection with a 3D hyperplane will result in four possibilities [3]:

1. **Nothing**: Occurs when the hyper-plane does not intersect the tetrahedron.
2. **One Vertex**: Occurs when the hyper-plane intersects the apex of a tetrahedron.
3. **Three Vertices**: Occurs when three edges intersect.
4. **Four Vertices**: Occurs when four edges intersect.

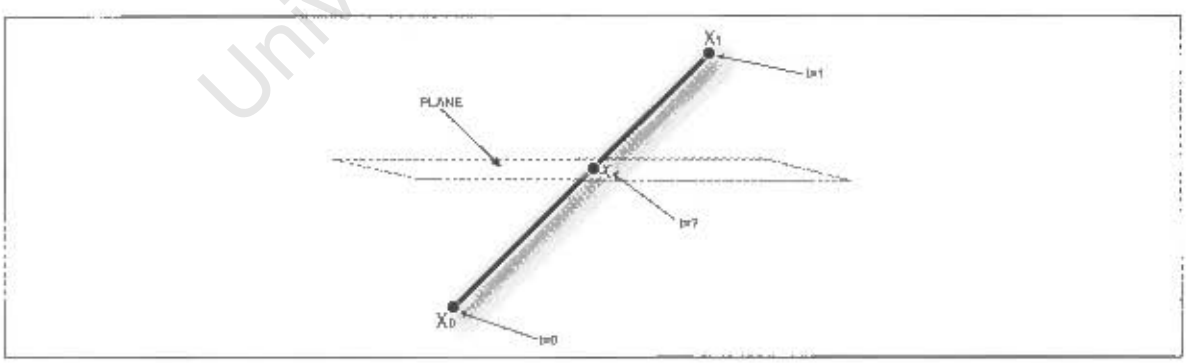
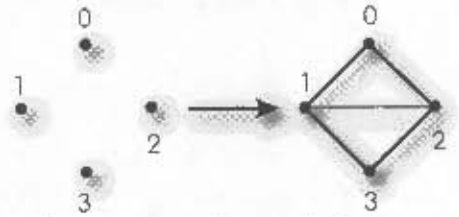


Figure 6.5: **Plane Edge Intersection:** A point on the line, when evaluated with the linear parametric equation of a line, has a specific value for t where the plane intersects the line. Note: X_0 occurs when $t = 0$, and X_1 occurs when $t = 1$.

Example 6.1 - Triangulation Of Convex Point Set Compared To Non-Convex Point Set

Assuming we have a convex point set consisting of four points $\{0, 1, 2, 3\}$, the aim is to produce triangulations whose union is identical to the convex hull of the point set:

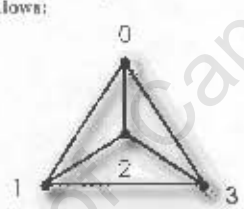


It is trivial to triangulated this point set: 4 points = 2 triangles:

- **Triangle 1**: $\{0, 1, 2\}$
- **Triangle 2**: $\{1, 3, 2\}$

This approach can be generalised for all point sets that contain four points and are convex. An example of a non-convex set:

Using the convex approach, an incorrect triangulation of the point set is produced. The correct triangulation would have been as follows:



Case 1 implies the tetrahedron is not involved in the extraction and can therefore be ignored. Case 2 is avoided with a slight oscillation of the vertex positions in w. Cases 3 and 4 are the expected outcome of a non-degenerate tetrahedron hyper-plane intersection. If three vertices are produced, triangulation happens immediately. If four vertices are produced, these vertices will lie on the convex hull, therefore it can be decomposed into two triangles with ease (Example 6.1).

The order in which the three vertices of a triangle are stored determines the ordering of the two vectors that define the 2D plane it is incident upon. A third vector, namely the surface normal can be calculated from these two vectors, which in turn can be used to determine if the face is back-facing. It is generally accepted practice that vertices ordered in an anti-clockwise fashion are defined to produce vectors whose normal will be front-facing. Triangle faces are therefore said to have an anti-clockwise winding. An extracted 3D model must produce windings that are identical to the 3D model used for extrusion. Therefore, if a triangle was back-facing in the original, the equivalent triangle must be back-facing in the extracted model. Because points are generated from 4D edges, the order in which the edges are traversed to produce the extracted vertices will determine the winding (Figure 6.6).

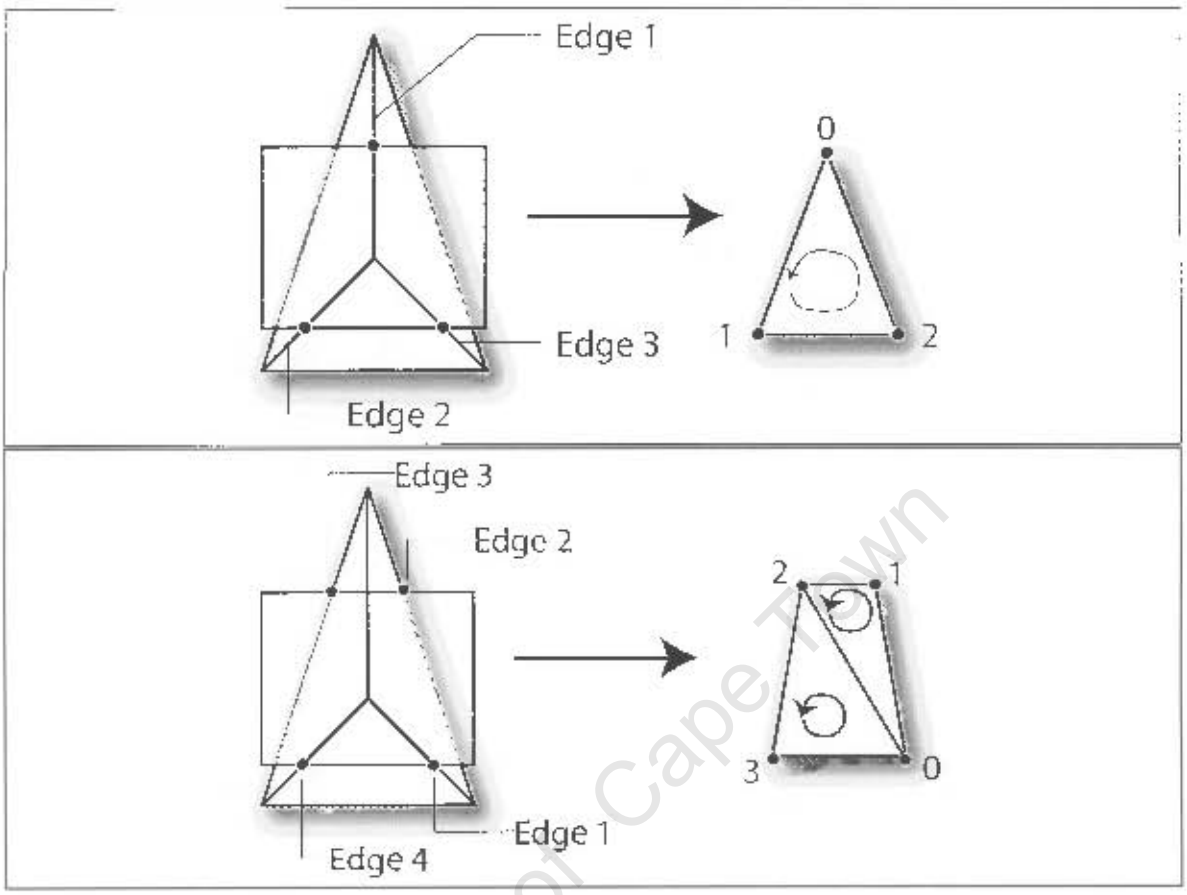


Figure 6.6: **Edge Traversal Determines Triangle Windings**: The order in which edges are traversed determines the windings of the resulting triangle. In the above diagrams, edges are traversed in numerical order.

Edges are traversed according to the ordering in Figure 4.10. Traversal of edges for extraction happens in the following order: Edge 4, Edge 1, Edge 5, Edge 2, Edge 6, Edge 3. This heuristic scheme ensures extracted vertices are ordered with correct windings.

If four vertices results from an extraction, there are two ways, each with anti-clockwise windings, in which the vertices can be triangulated (Figure 6.7). This is because the four vertices are aligned in the 3D hyper-plane, but not necessarily in the 2D plane. The best triangulation scheme is a Delaunay Triangulation, since this minimises the maximum angle over the two triangulations [39]. The four points lie on a convex hull, thereby making both triangulation schemes correct, but not necessarily optimal. A decision is taken for the sake of efficiency to triangulate consistently rather than optimally resulting in a heuristic for each of the six prisms of Figure 4.15 that are *always* triangulated in a consistent manner whenever an extraction results in four points.

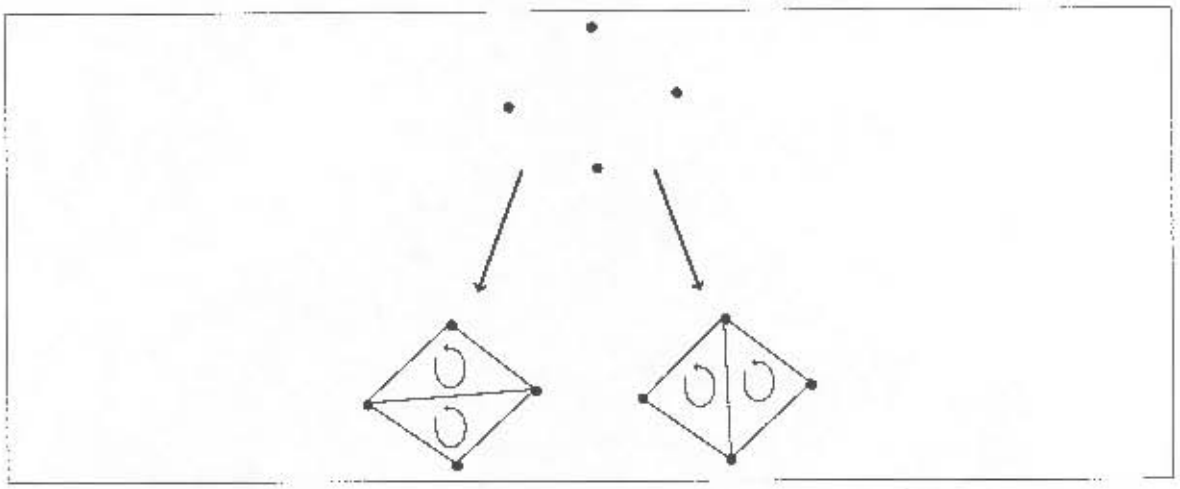


Figure 6.7: **Four Point Triangulation**: There are two correct ways to triangulate four points.

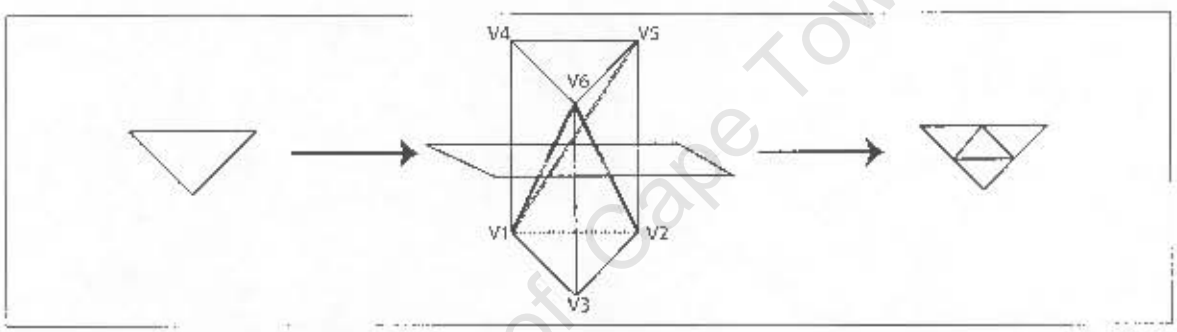


Figure 6.8: **Hyper-Plane Tetrahedronalised-Prism Intersection** Prisms result from connecting the same triangle in corresponding levels. The result of intersecting a hyper-plane with a tetrahedronalised prism will produce four triangles

Intersection Of Self-intersecting Prisms

One of the problems with this approach is that the windings become incorrect if a post-deformation prism self intersects. A test to determine self intersection before deformation in order to correct the problem is needed. This problem is not solved in this dissertation (Refer to Chapter 8 Section 8.2 for future work) as it is outside its scope. It is, however, essential in order to obtain a correctly working model.

Result Of A Hyper-Plane Tetrahedronalised-Prism Intersection

Intersecting a tetrahedronalised prism with a hyper-plane results in a point set that produces four triangles. Prisms result from connecting triangles in corresponding levels of the 4D object (Section 4.1), therefore prism extraction produces four triangles as output where only one is used as input (Figure 6.8).

6.1.3 Production Of Normal Information For The Extracted Model

Due to conservation of space, no normal information is stored in the 4D model. If normal information were stored in the 4D model, it would need updating after a deformation, which would hinder interactivity since a Jacobian calculation would be required [26]. The extracted 3D model needs normal information for rendering. Exact normal information produced by using a Jacobian is deemed unnecessary for rendering, instead normal information is calculated by averaging normals for all faces incident on a vertex. If this were to be done for every face it would hinder interactivity, but optimisations (Section 6.2) reduce the set of vertices needing normal recalculation to a subset consisting of vertices involved in the 4D deformation.

6.2 Efficiencies

6.2.1 Efficiencies At The Edge Level

Extracted Vertex Storage

A reference to the extracted vertex is stored in the edge where it was produced after the first extraction query. All subsequent extraction queries are immediately handed the reference thereby avoiding hyper-plane intersection. An edge that has been involved in an extraction will only be used again if it is shared by another tetrahedron. In this case, if the edge is intersected with the hyper-plane again, two extracted points with the same coordinates would be produced. Instead extra calculations are avoided and a viable result is produced.

Hyper-plane Calculation

The production of a 3D hyper-plane involves using the 4D determinant in order to determine the normal of the hyper-plane. This is computationally costly, and it may be the case that the user would want to extract more than one 3D model. If this is the case, the nature of extrusion construction (Section 4.1.2) guarantees the 4D model will always be aligned to the fourth dimension. Therefore to produce a 3D hyper-plane for extraction, the normal $(1, 0, 0, 0)^T$ is all that is needed. The point on the hyper-plane, a user determined value, is the only attribute that needs adjusting.

6.2.2 Efficiencies At The Tetrahedral Level

Tetrahedral-Edge Intersection

Intersection of a hyper-plane and a tetrahedron will result in at most four points (Section 6.1.2). A tetrahedron has six edges, but not all edges need be tested; testing is only performed until four extractions have been produced. This will only be efficient for tetrahedra that produce four points,

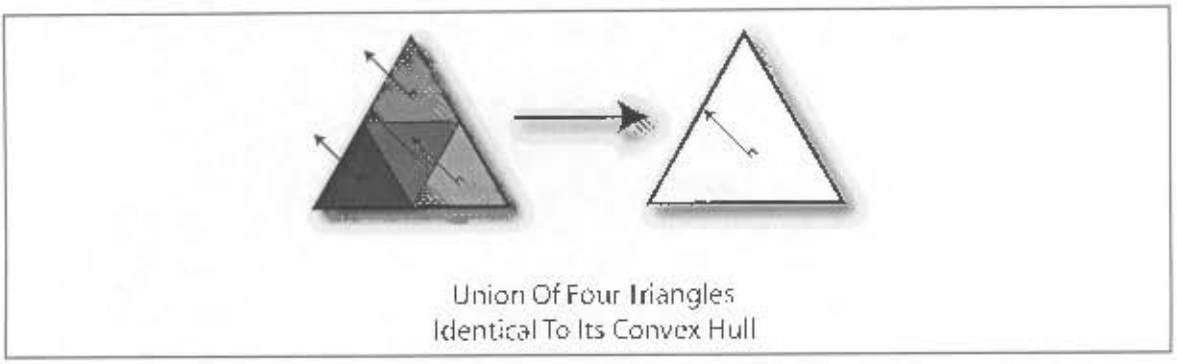


Figure 6.9: **Four Extracted Triangles Identical To Its Convex Hull:** The result of triangulating extracted points from an undeformed tetrahedronalised prism produces four triangles that are identical to its convex hull. The convex hull in turn is identical to the triangles that constitute the undeformed prism. Therefore, these four triangles can be substituted with one of the triangles that make up the prism

since tetrahedra that produce three points will still check all edges for a potential extraction.

Prism Intersection

A tetrahedronalised prism intersected with a hyper-plane will result in an extracted point set that produces four triangles (Section 6.1.2). If the prism is undeformed, the union of these four triangles will be identical to its convex hull. That is, all four triangles will have identical normals and their union will be coterminous to the triangles that comprise the undeformed prism (Figure 6.9). It makes no sense to extract an undeformed prism, therefore, the extraction algorithm returns the original triangle used to build the undeformed 4D prism. Hyper-plane edge intersection and triangulation are avoided resulting in less triangles. Normal information is not recalculated since it stays the same.

This approach is problematic when an undeformed prism neighbours a deformed prism: A deformed prism will produce four triangles, whereas an undeformed prism will produce one triangle. This results in the undeformed triangle having an edge with two neighbours incident upon it and the model is no longer a simplicial complex (Section 3.2.2) (Figure 6.10). This is repaired by splitting the undeformed triangle in two (Figure 6.10). Vertices do not need their normals updated, and the two 'children' triangles will have identical normals to their parent.

A deformed prism will have an undeformed neighbour provided only one vertical edge has been deformed (Figure 6.11). Thus, a deformed prism can only ever have *one* undeformed neighbour.

The deformation process flags tetrahedra that have been deformed (Section 5.3.2). To determine if an undeformed prism neighbours a deformed prism, the neighbour equation (Equation 4.1) is used. Not all nine tetrahedral neighbours need be examined since tetrahedra in clumps of three (prisms) share vertices; therefore if one tetrahedron in a clump of three has not been deformed, the whole prism is undeformed. If a neighbour is undeformed, the model will be in a non simplicial complex state and will need repairing. Splitting the undeformed triangle into two involves two operations:

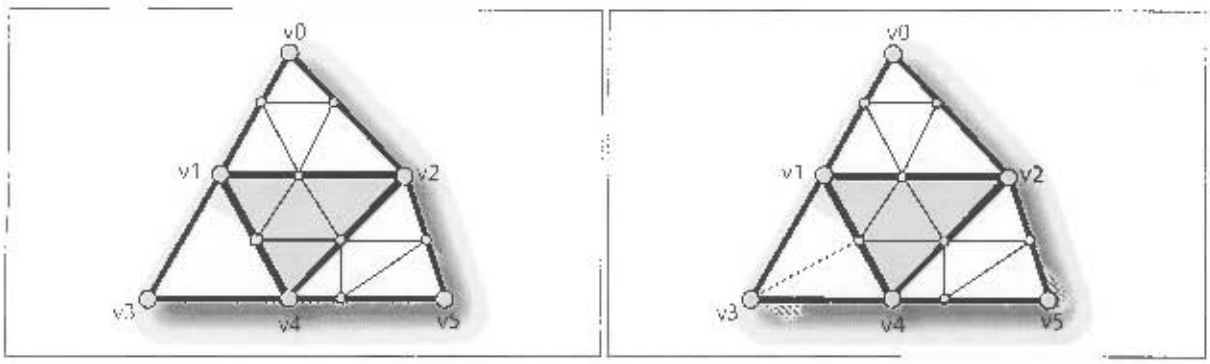


Figure 6.10: **Consistent Triangulation After Deformation:** [Left] The vertex v_2 has been deformed with prisms that embody the vertex also being deformed resulting in extractions of four triangles. The left most prism does not embody v_2 therefore producing an undeformed prism with a one triangle extraction. This results in a non-simplicial complex with the left-most triangle having an edge with two neighbours incident upon it. The vertex that is present along the non-conforming edge is drawn as a square [Right] The model is corrected by splitting the undeformed triangle into two triangles.

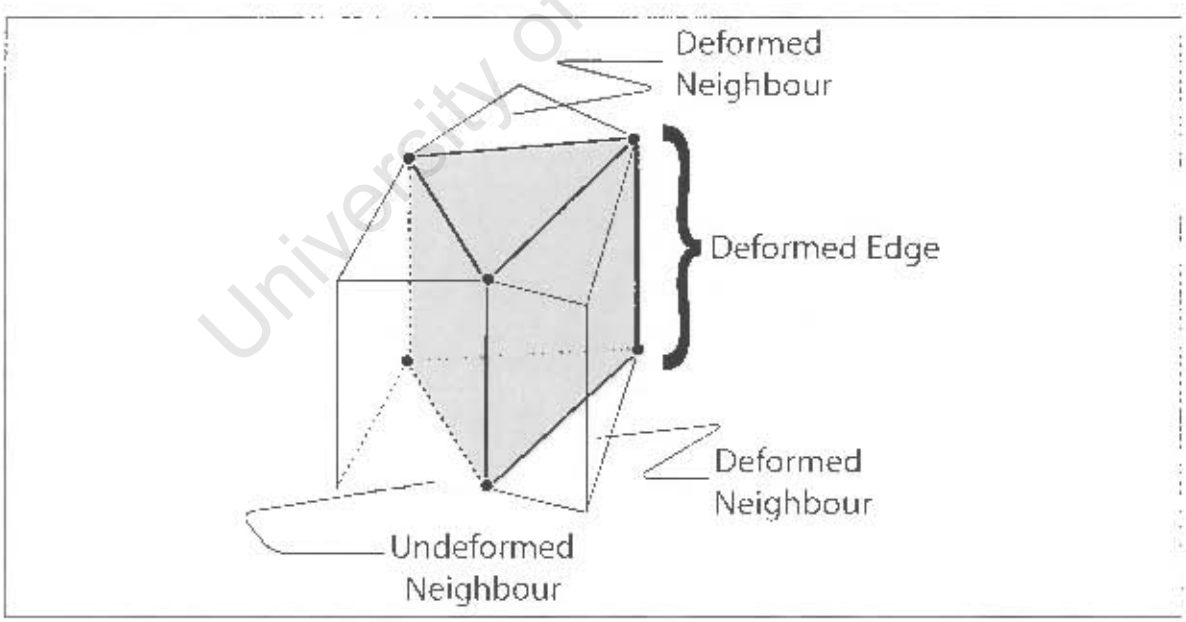


Figure 6.11: **Deformed Prism Neighbours**: If a deformed prism has an undeformed prism neighbour, then at most only one vertical edge has been deformed.

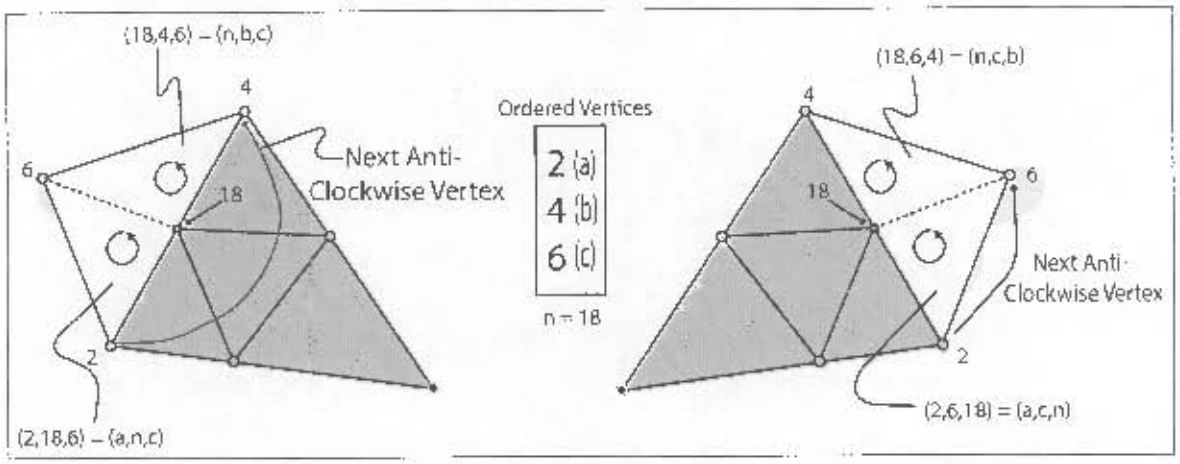


Figure 6.12: **Splitting The Undeformed Triangle With Correct Windings**: [Left] The next anti-clockwise vertex is not on the shared edge - triangles are (a,n,c) (n,b,c) [Right] The next anti-clockwise vertex is on the shared edge - triangles are (a,c,n) (n,c,b)

- **New Vertex**: The vertex *on* the edge that has two neighbours must be determined. This vertex is not part of the edge, and therefore causes the non-conformity (Figure 6.10).
- **Triangle Winding**: Split triangles must have the same winding as the original triangle.

Determining the vertex index along the non-conforming edge depends upon which one of the three prism neighbours were undeformed, and which of the two cases for tetrahedral subdivision the prism belongs to. Using the edges of the second of the three tetrahedra[1] derived from the deformed prism, the vertex is:

- v_2-less:
  - **Neighbour 1 Undeformed**: Extracted from edge 1.
  - **Neighbour 2 Undeformed**: Extracted from edge 4.
  - **Neighbour 3 Undeformed**: Extracted from edge 6.
- v_3-less:
  - **Neighbour 1 Undeformed**: Extracted from edge 4.
  - **Neighbour 2 Undeformed**: Extracted from edge 1.
  - **Neighbour 3 Undeformed**: Extracted from edge 2.

Once the new vertex index has been determined, it must be used to split the undeformed triangle in two with appropriate windings. This is accomplished in four steps (Figure 6.14 depicts the first two steps, and Figure 6.15 the last two steps):

[1] Using the second tetrahedra is an arbitrary choice.

- **Determine The Triangle That Produced The Deformed Prism And Order Its Vertices**: The triangle in the 3D model that produced the deformed prism is determined by dividing its index with three using integer division. Its vertices are ordered (assume ordered vertices are (x, y, z))and associated with its neighbours in the following way(Section 4.1.1): If the undeformed neighbour is neighbour 1, the edge is (x, y), neighbour 2 (x, z) and neighbour 3 (y, z).

- **Determine The Position Of The Least Vertex Of The Edge In The Undeformed Triangle**: The position of the least index of the edge shared by the undeformed triangle is determined in the undeformed triangle's vertex array.

- **Determine The Next Anti-Clockwise Vertex**: The next anti-clockwise vertex is determined. The triangle vertex array hold three indices referencing its three vertices in anti-clockwise order. If the position of the current vertex of interest in the array is x, the position of the next anti-clockwise vertex is $(x + 1) mod\ 3$

- **Split Undeformed Triangle In Extracted Model**: The undeformed triangle is split in two with appropriate windings. Assume the ordered indices of the triangle are (a, b, c) and the new vertex is n. Triangle creation is divided in two cases, depending upon the orientation of the undeformed neighbour (Figure 6.12):

  - **Case 1**: The next anti clockwise index $((x + 1) mod\ 3)$ is the other vertex in the shared edge. The two triangle faces are created with the following vertices (Figure 6.12 - left):
    * (a, n, c)
    * (n, b, c)
  - **Case 2**: The next anti clockwise index $((x + 1) mod\ 3)$ is *not* other vertex in the shared edge. The two triangle faces created with the following vertices (Figure 6.12 - right):
    * (a, c, n)
    * (n, c, b)

6.2.3 Vertical Edge Extraction

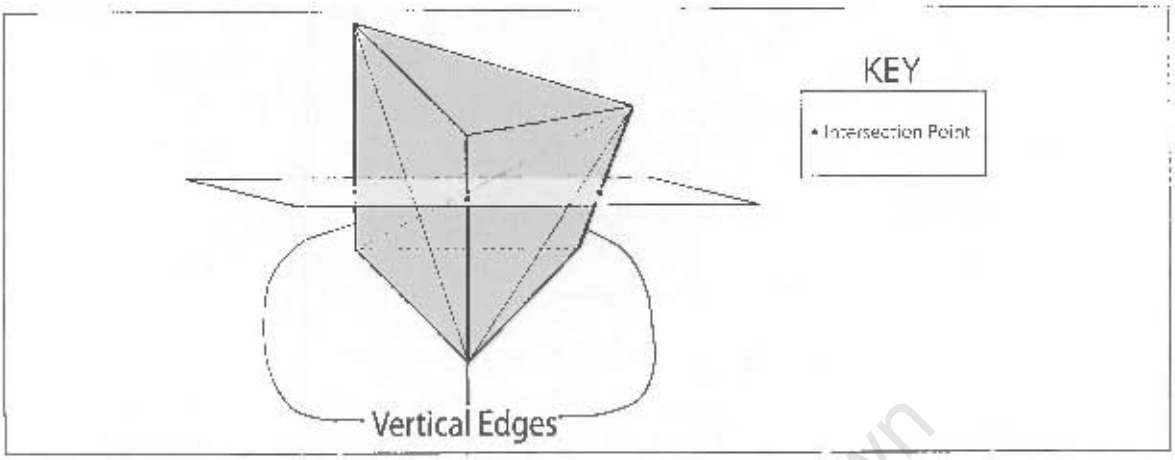


Figure 6.13: **Deformed Prism Vertical Edge Intersection**: An intersection of a deformed prism that encompasses its vertical edges need not have its tetrahedra extracted.

A full extraction is not always required for a deformed prism. If the hyper-plane intersects a deformed prism's three vertical edges and does not intersect its base triangles (Figure 6.13), it is not necessary to intersect its subdivided tetrahedra. A connection scheme will potentially be ambiguous only if edges making up the triangles of the prism are intersected. It is then necessary to involve the tetrahedra in the extraction. The result of intersection will be a single triangle as opposed to four triangles.

Since the prism has been deformed, edges have to undergo extraction in order to produce the deformed coordinates. There are three vertical edges, resulting in three extracted points which can immediately be triangulated. A potential problem arises when a prism whose vertical edges intersects a neighbouring hyper-plane prism whose edges do not. The problem is resolved identically to Section 6.2.2, with triangles being split in two. Prisms whose vertical edges undergo intersection are flagged as *vertical*, while prisms that need tetrahedral intersection are flagged as *tetrahedral*. During prism intersection, prisms neighbours are checked for the following flags:

- **Vertical**: If the current prism is tetrahedral, a correction must be made.
- **Undeformed**: Correction is needed if the current prism is tetrahedral.
- **Tetrahedral**: Corrections are needed if the current prism is undeformed or vertical.
- **Flag Not Set**: The prism is ignored and no corrections are made. When the intersection algorithm cycles through the prism, its flag will be set, and a suitable check will be made.

Table 6.1 summarises the first of the above three items. If at least one prism's neighbour flag has not been set, no decision can be made to correct the extracted 3D model. In this case, the decision

is deferred until the flag is set (done just after the prism has been extracted). A Decision can only be made once all a prisms neighbours have had their flags set.

Prism Type	**Undeformed**	**Vertical**	**Tetrahedral**
Undeformed	No Correction	No Correction	Correction
Vertical	No Correction	No Correction	Correction
Tetrahedral	Correction	Correction	No Correction

Table 6.1: **Prism Correction Classification**: A cell in column x and row y means the prism with classification in column x neighbours the prism with classification in row y. This table depicts the cases when correction to the extracted model will be necessary.

6.2.4 Efficiencies At The Model Level

Exact Level Calculation

A hyper-plane intersects a 4D model at a particular level. This level will contain most of the tetrahedra, barring the deformed tetrahedra, whose edges will be intersected. An exact and efficient calculation to determine the start and end indices of all tetrahedra existent upon a level can eliminate many unnecessary checks to determine if prism intersection occurs.

The index of the start tetrahedra in a particular level is the level number multiplied by the number of tetrahedra per level. The index of the last tetrahedra is the number of the next level, multiplied by the number of tetrahedra per level, with one subtraction. Due to coherence, the indices of all tetrahedra between the start and end are all the tetrahedra present in that level.

Since deformed tetrahedra may not be aligned to other levels, exact extraction will also have to examine all deformed tetrahedra. Tetrahedral intersection tests are therefore restricted to deformed tetrahedra.

Deformed Tetrahedra Extraction

Extraction of an undeformed 4D object will produce the original 3D object. It is the deformed regions of the 4D object that produces the topology alteration. Deformed tetrahedra are examined, and extracted if an intersection occurs. This results in a subset of the extracted model, therefore the remainder of the extracted model must be built. This can be achieved in two ways:

- **Implicit Use Of 3D Model**: The original 3D model is altered by appending information produced from the deformed extractions. Due to the redundancy of information to the original 3D model, structure from non-deformed areas of the mesh do not need to be explicitly built.
- **Exact Deformed Extraction**: The extracted model is built by an exact extraction appended to a deformed extraction.

An implicit use of the 3D model is the preferred means for extractions, but altering the original model implies that another extraction will not be possible. If more than one extraction would be required, then exact deformed extraction is be more viable.

6.3 Summary

The extraction algorithm has been described. Extraction is performed by intersecting a hyper-plane with the extruded model. This results in a point set which is then triangulated. Efficiencies aimed at speeding up extraction are introduced. Efficiencies pertain to a three levelled extraction hierarchy; (1)the model, (2)the tetrahedra and (3)the edges. Efficiencies aim to reduce unneeded operations and decrease the number of resulting triangles.

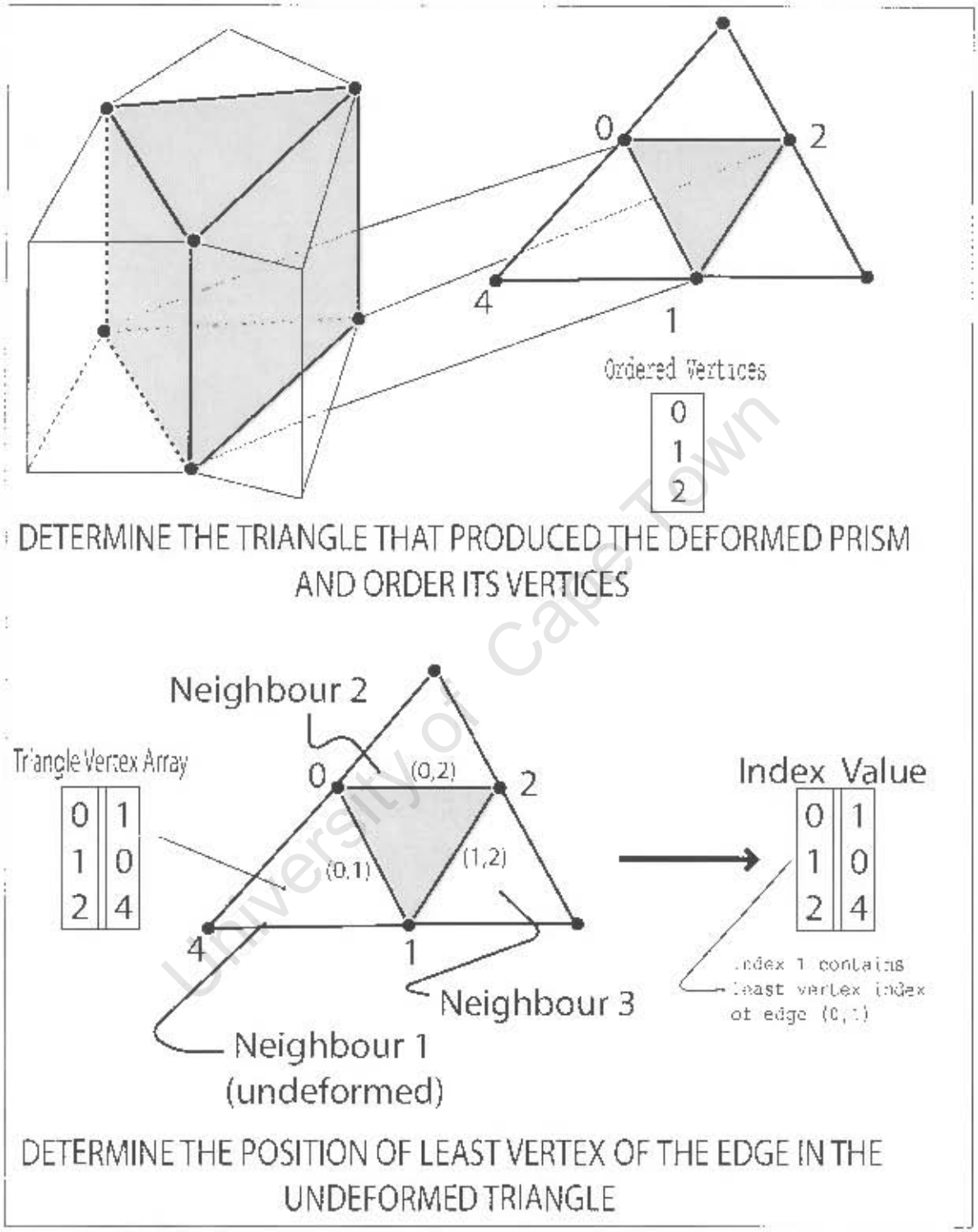


Figure 6.14: **Making The Extracted Mesh Consistent:** The first two steps for making an extracted mesh consistent when an undeformed triangle neighbours deformed extracted triangles.

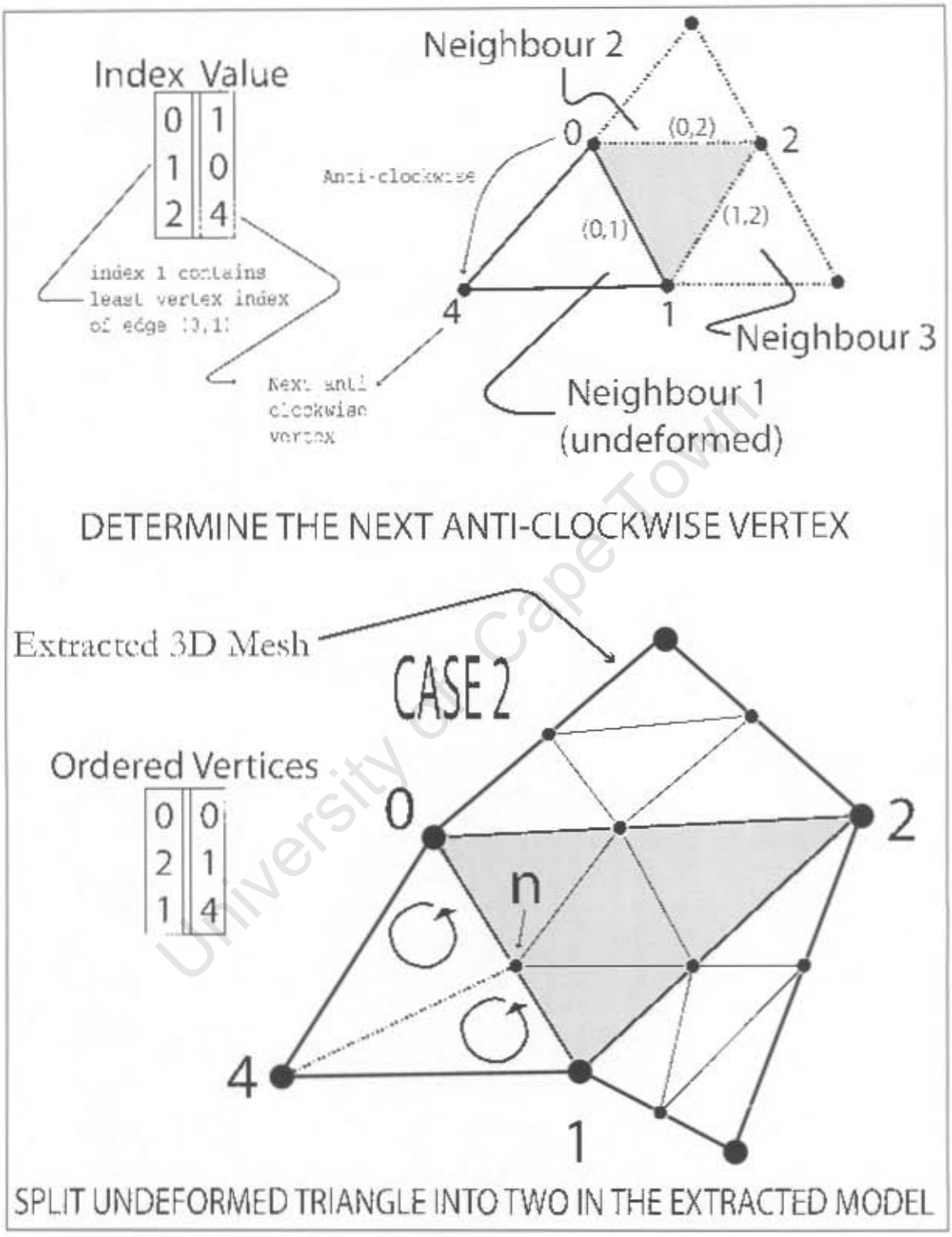


Figure 6.15: **Making The Extracted Mesh Consistent**: The last two steps for making an extracted mesh consistent when an undeformed triangle neighbours deformed extracted triangles.

Chapter 7

Results and Evaluation

7.1 Testing Criteria

Results are obtained as an average of 100 timings. The visual c++ profiler was avoided due its erratic nature, instead a high resolution timer function was used with timings captured in milliseconds. Results were obtained on a PC with the following specifications:

- **Processor**: Dual Pentium 4 *1.68*GHz.
- **Ram**: *2.25*GB.
- **Video Card**: Geforce 2 MMX with *64*MB on board ram.
- **Operating System**: Microsoft Windows XP.
- **Development Platform**: Microsoft Visual C++, version 6.0.

Interactivity is defined as 15 updates per second. In other words, for a topology alteration to be interactive, it cannot take longer than 67 milliseconds.

7.1.1 Models

Ten models were chosen to test topology alteration, See Table 7.1 for a pictorial description of the models. Models chosen differed in vertex and polyhedron count, with the Stanford Bunny having the highest vertex and polygon count. The models are:

- **Cube**: A low polygon and vertex count with a simple topology. It was used throughout the development stage to test the algorithm.
- **Dolphin**: A relatively low polygon and vertex count, the dolphin is typical of a polygon that might have been produced using virtual sculpting.

- **X-Wing**: A more detailed polygon mesh, it represents a model that may have been laser scanned, and then substantially simplified.
- **Rifle**: The rifle has a complex topology: The trigger protector gives it genus 1.
- **Bass Guitar**: More complex than the rifle, but with a simple topology.
- **The Sting Ray**: The string ray has a complex topology with genus 1.
- **Mondo Man**: Mondo man's head floats above his body. The portion where his legs are attached to his torso is also genus 1.
- **Cow**: The traditional Stanford cow which has been simplified.
- **Sphere**: A large sphere.
- **Bunny**: The Stanford bunny used to test limits of the topology alteration algorithm. It was also used to test if models fitted the linear regression. As it is so much bigger than the previous models, it is left out of the graphs.

Cube Number Of Vertices: *8* Number Of Triangles: *12* Number Of Edges: *20*		
Dolphin Number Of Vertices: *469* Number Of Triangles: *937* Number Of Edges: *704*		
X-Wing Number Of Vertices: *775* Number Of Triangles: *1289* Number Of Edges: *2062*		
Rifle Number Of Vertices: *1119* Number Of Triangles: *1845* Number Of Edges: *2962*		

Bass Guitar Number Of Vertices: *1419* Number Of Triangles: *2253* Number Of Edges: *3670*		
The Sting Ray Number Of Vertices: *2928* Number Of Triangles: *3753* Number Of Edges: *6680*		
Mondo Man Number Of Vertices: *2178* Number Of Triangles: *4084* Number Of Edges: *6260*		
Cow Number Of Vertices: *2904* Number Of Triangles: *5804* Number Of Edges: *8706*		
Sphere Number Of Vertices: *4098* Number Of Triangles: *8192* Number Of Edges: *12288*		
Bunny Number Of Vertices: *35947* Number Of Triangles: *69451* Number Of Edges: *105396*		

Table 7.1: **Models**: The models and their respective vertex and triangle count used to determine results.

7.2 Results

7.2.1 Extrusion

Extrusion timings are dependant upon the most numerous components of the model; the edges. This is due to 4D edges being computed and stored for extraction. Unfortunately, there are more edges than vertices or faces (Euler's law). Timing results (Figure 7.1) were fitted using a linear regression, with the Stanford Bunny being tested to confirm the result. Extrusion optimisations improves timing results by a factor of 2. The optimisations prevent unnecessary calculations after the computation of the first level, therefore a factor of more than two was expected. The bottleneck is due to the standard template library (STL) vector class as it is implemented in Microsoft Visual C++ 6.0: A call to reserve memory is not as efficient as on another platform. This was only discovered after implementation.

7.2.2 Deformation

Optimising the embedding algorithm produces significantly improved results as demonstrated by Figure 7.2. All models tested in Figure 7.2 were extruded using ten levels. Optimised embedding results on average in an improvement factor of 18. Optimisations use embedding information from the 3D model, the only calculation being the 4D component, which needs to be calculated just once per level. Embedded coordinates for the 4D model are calculated just before deformation. In addition, optimisations introduced in [26] and adapted to 4D aid the interactivity of the deformation process.

7.2.3 Extraction

Extractions are tested after deforming roughly eighty percent of the specified cells for each model per level (Figure 7.3). A linear regression is fitted to the data and tested against the Stanford Bunny. In order to test optimisations, the sphere is extracted with deformations that span different proportions (Figure 7.4). Figure 7.4 shows that extracting only deformed faces is an improvement over extracting the whole object. Timings are logarithmic due to edges storing results from previous extractions being used in subsequent calculations.

7.2.4 Overall

Timings for a deformation that results in a topology alteration after extraction are presented (Figure 7.5). Interactivity is breached if times exceeds 67 milliseconds. Results show that it is a model's triangle count, and not its vertices, that determines the time for topology alteration. The cow has less vertices than The Sting Ray, yet its triangle count is much larger, resulting in longer overall topology alteration time.

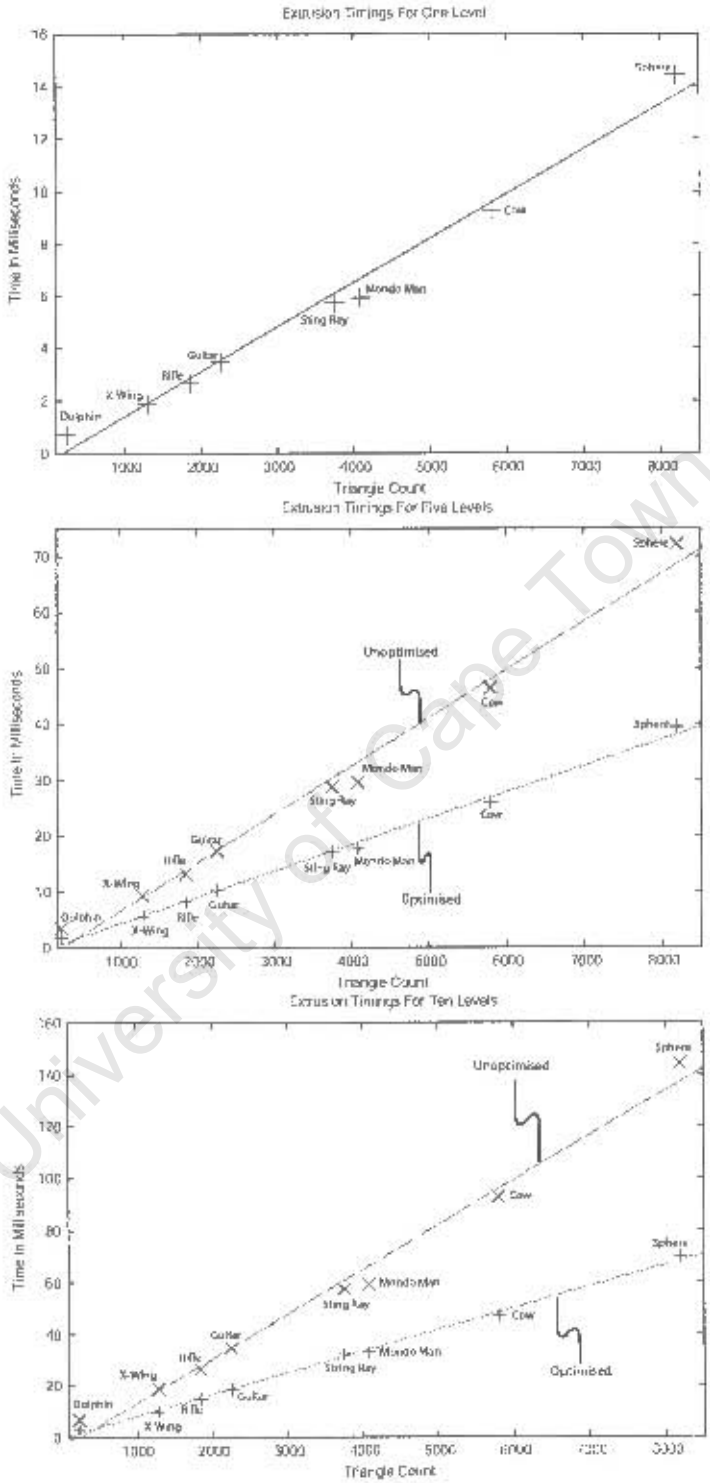


Figure 7.1: **Extrusion Timings**: Graphs representing extrusion timings [Top] Extrusion timings for one level [Middle] Extrusion timings for five levels [Bottom] Extrusion timings for ten levels.

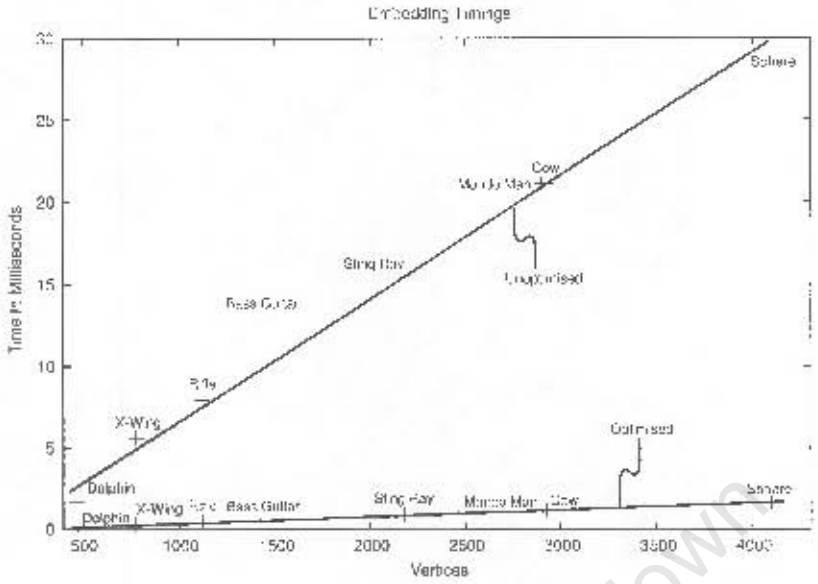


Figure 7.2: **Lattice Embedding Timings:** Unoptimised embedding times VS its optimised counterpart.

Extrusion timings rely not so much on the amount of levels chosen as on the complexity of the model. Even though performance degrades with more extrusion levels, it does not require many levels to perform a topology alteration. Also due to optimisations, the first level of extrusion is the most performance demanding step, while redundancy dramatically reduces the time needed to extrude the other levels. Optimisation also aids in the deformation and extraction performance of extruded objects. Lastly, it is in the interests of memory conservation to keep the extruded levels to a minimum.

The amount of levels needed is deformation and model complexity dependant. An extruded object that has a large portion altered due to deformation would be a candidate for an increase in extrusion levels, thereby potentially reducing the altered portions. The preceding sentence uses the word 'potentially', since the amount an object is altered also depends on the deformation. A level number of five was used to render the picures for this chapter. Five levels of extrusion performed as well as ten for all models. Less than five levels produced warping artefacts for some of the models that should not be present in this technique, such as too much deformation along the x,y or z axises. For larger models such as the Stanford Bunny, less than five levels produced suitable results. This is because a less sparse (finer) hyper patch works better with lots of vertices, and produces more local warping effects. The number of levels chosen for an extruded object is therefore dependent on the complexity of the model. The more complex the model, the less the number of extrusion levels needed.

The topology alteration process presented in this disseration has one serious flaw: It is not able to combine objects. A 4D modeller based upon [12] is able to produce desired results. These were not

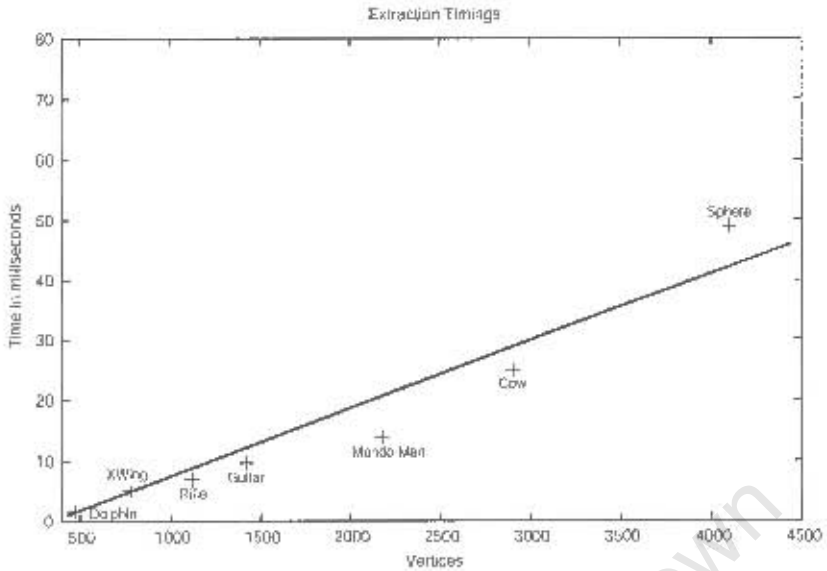


Figure 7.3: **Extraction Timings:** Timings for the extraction process.

however persued for this project as it was thought to be outside the scope of the project.

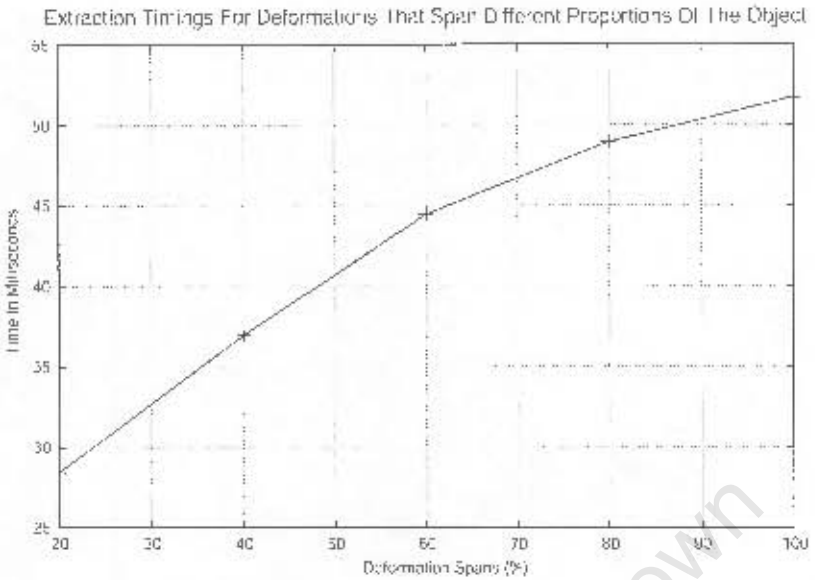
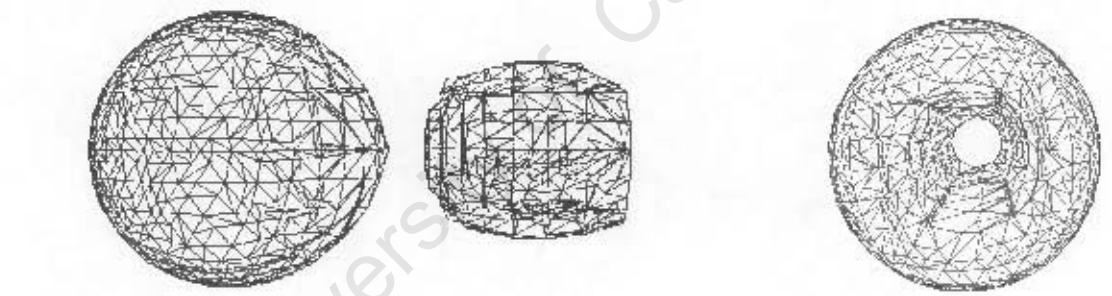
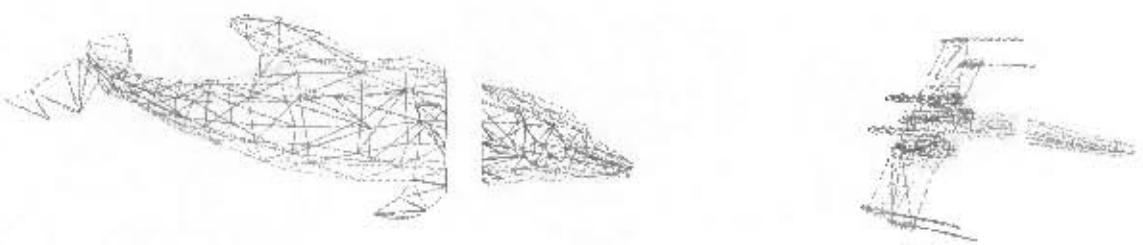


Figure 7.4: **Extraction Timings For Different Deformation Spans Of a Sphere**

7.3 Topology Alteration Pictures





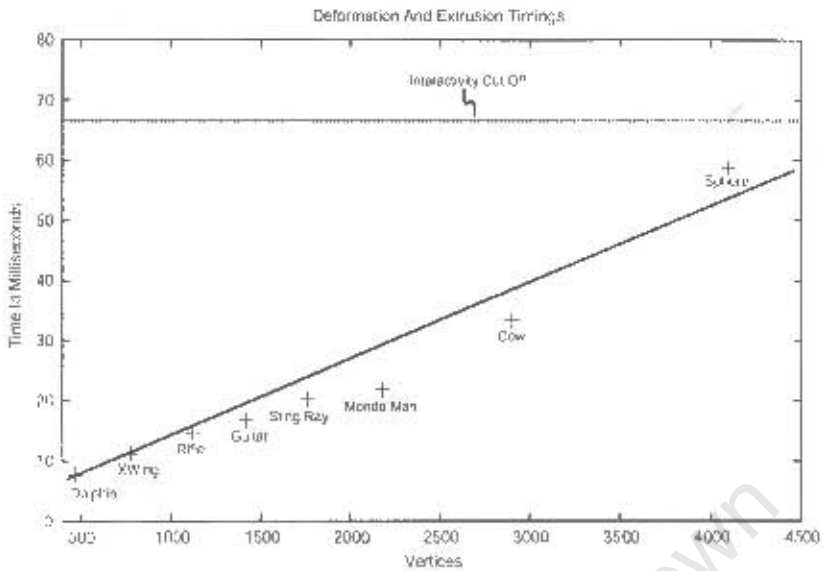


Figure 7.5: **Overall Topology Alteration Timings:** Timing results for the topology alteration process as a whole.

The above pictures all had five extrusion levels. The sphere hole was produced by a vector originating from its x,y,z origins with its w axis orginating from the fifth extruded level. The deformation vector kept the x,y and z parts the same, by emanated towars the first level in the w-axis. For all object separations, the x,y,z origins were at the exact place where the separation was wanted, with the w coordinate being the first level. Depending upon the orientation of the model, the resulting vector's x,y,z components changed, while the w component always headed towards the fourth level.

Deformation performance is dependant upon the type of deformation used. The reason being is that a deformation which results in a large proportion of the object being deformed will require a more thourough extraction and produce more resulting triangles. A more thourough extraction implies that the more prisms that are deformed, the more tetrahedra need to be extraced and prism neighbours need checking for non-deformed neighbours.

Chapter 8

Conclusion and Future Work

The focus of this dissertation is on producing an efficient means of altering a 3D model's topology under virtual sculpting implemented using spatial deformation.

Ideas proposed by Aubert and Bechman in [4] were extended to fit requirements needed for virtual sculpting. Their approach was chosen because it altered topology with most of the topological requirements posed in Chapter 1, with the exception of interactivity. Extensions therefore aim to make an unambiguous topology alteration process interactive.

The topology alteration process is split into three sub-processes: (1) *Extrusion*, (2) *Deformation* and (3) *Extraction*. A 3D object is extruded into 4D, after which, the 4D object is deformed and then intersected with a 3D hyper-plane, thus producing a 3D extracted object with potentially altered topology.

The extrusion process embeds a 3D object into multiple 4D layers, each layer having a unique 4th coordinate. Vertices of triangle faces in the 3D model are identified in 4D and are connected in consecutive 4D levels producing 4D prisms. Prisms are then subdivided into tetrahedra resulting in connectivity information that is a 3-simplex.

3D Direct Manipulation of Free-Form Deformation (DMFFD) (Section 3.1.3) is adapted to 4D for deformation of the extruded object.

A 3D hyper-plane is intersected with the 4D extruded model to produce a point set which is then triangulated. Due to 4D tetrahedral connectivity, the point set can be triangulated unambiguously. Depending on the deformation specified, the extraction produces an object with altered topology.

8.1 Optimisations

Optimisations to the extrusion process (Chapter 4) involve exploiting spatial coherence produced by the embedding process. Edge and tetrahedral computation benefit from coherence. All subsequent tetrahedronalisations and edge calculations after the first level exploit coherence for their calculations.

Extrusion time is linearly dependent upon the number of edges in a model. Therefore extrusion time increases with model complexity. This does not hinder interactivity as extrusion is designed to be a pre-process performed in conjunction with the loading of the 3D model. The Stanford bunny, considered to be a large model for virtual sculpting, takes 580ms for a ten levelled extrusion. Optimised extrusion results in a speed up by a factor of two.

4D deformation (Chapter 5) builds upon optimisations produced by Gain in [25]. The most important of which is an optimised means for the calculation of a pseudo-inverse (Section 3.1.4). The optimised embedding algorithm uses embedding information pertaining to the 3D model. It results in a speed up by a factor of 18.

Extraction optimisation only intersect tetrahedral faces of the 4D object that were deformed. The resulting triangulated point set is appended to the original model.

The topology alteration method can achieve a deformation and extraction in interactive time (15 frames a seconds), with the extrusion as a pre-process. Although hole creation and surface separation can be achieved, surface joining is not possible. This is due to the nature of the space-time object, which is not adept at object joining. Self intersecting of tetrahedra result in triangles with incorrect windings and produces models with 'holes'.

8.2 Future Work

There are several areas for future research which pertain to this dissertation:

- **Refinement/Decimation**: A refinement/decimation scheme would eliminate errors due to sampling. The curvature of the 4D model may be degraded after deformation, resulting in an extraction that does not smoothly approximate the curvature. This can be remedied, to some extent, by sampling the extrusion more densely resulting in a 4D object with more levels closer to each other. A refinement scheme makes this unnecessary, allowing for a relatively small extruded object. A decimation scheme will eliminate samplings that are too dense, thus reducing unnecessary detail.

- **Self Intersection Test**: A problem with extraction is that tetrahedra may become self intersected after deformation resulting in them being turned inside out. This produces point sets that get triangulated with incorrect windings. A self intersection test has been developed for 3D virtual sculpting in [28]. Adapting this test to 4D will eliminate the tetrahedra self

intersection problem.

- **Surface Joining**: Deforming a higher dimensional object and then extracting a potentially topologically altered object will not be able to create a joining of surfaces. Either attempting to adapt this method for surface joining, or using another scheme in a hybrid combination would be interesting future research.

Appendix A

Geometry

A.1 Plane-Line Intersection

A plane is defined by a normal $\vec{N}$ and a point P on the plane, while an edge is defined as two vertices. To intersect an edge [1] with a plane, we require the following variables: (Figure A.1):

- P - A point on the plane.
- $\vec{N}$ - A normal to the plane.
- P_1 - The starting vertex of an edge that intersects the plane.
- P_2 - The end vertex of an edge that intersects the plane.

We find a solution for t such that the following parametric equation for a line will yield a point P_3 on the plane:

$$P_3 = y(t) = (1-t)P_1 + tP_2,\ t \in [0,1] \tag{A.1}$$

Since P_3 is assumed to be on the plane, the dot product of the vector from P to P_3 and $\vec{N}$ will be zero:

$$\vec{N} \cdot (P_3 - P) = 0 \tag{A.2}$$

Rewriting equation A.1 to $P_3 = y(t) = P_1 + t(P_2 - P_1)$, and substituting this for P_3 in equation A.2, we get:

$$\vec{N} \cdot (P_1 + t(P_2 - P_1) - P) = 0 \tag{A.3}$$

[1] Note: We use the terms edge and lines interchangeably here.

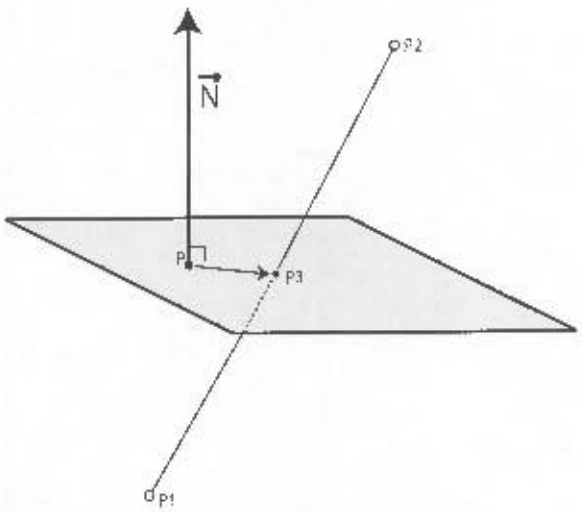


Figure A.1: **Plane-Line Intersection**

Solving for t gives:

$$\begin{aligned} \vec{N} \cdot (t(P_2 - P_1)) &= \vec{N} \cdot (P - P_1) \\ t(\vec{N} \cdot (P_2 - P_1)) &= \vec{N} \cdot (P - P_1) \\ t &= \frac{\vec{N} \cdot (P - P_1)}{\vec{N} \cdot (P_2 - P_1)} \end{aligned} \tag{A.4}$$

Substituting t of equation A.4 into equation A.1 yields the desired result. The following code is used to produce plane-line intersections:

```
Point planeLineIntersect(Point startEdgeVertex, Point endEdgeVertex) {
    Vector v1 = pointOnPlane - startEdgeVertex;
    Vector v2 = endEdgeVertex - startEdgeVertex;
    float t = v1.dotproduct(planeNormal) / v2.dotproduct(planeNormal);
    return (1-t)*startEdgeVertex + t*endEdgeVertex;
}
```

Appendix B

Linear Algebra

B.1 Proofs for Equations 5.3

The proof of how s is derived is given here. The other coordinates, namely t and u follow trivially.

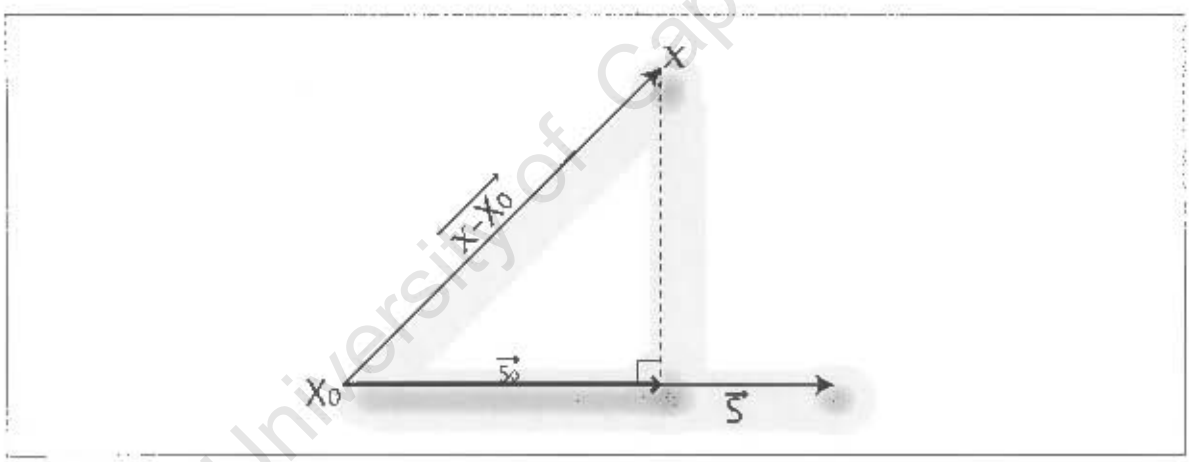


Figure B.1: **Projection Of $\mathcal{X}$ Onto $\vec{S}$:** To determine the local coordinates of $\mathcal{X}$ in the $\vec{S}$ direction, $\mathcal{X}$ must be projected onto $\vec{S}$ to produce $\vec{s_0}$

From the above figure, it can be seen that s is the projection of $\vec{s_0}$ onto $\vec{S}$:

$$s = \frac{\| \vec{s_0} \|}{|| \vec{S} ||}$$

Assuming $\vec{S}, \vec{T}, \vec{U}$ of Section 5.1.1 are not orthogonal, Equation 5.2 need be used to determine s:

$$s = \frac{(\vec{T} \times \vec{U}) \cdot (\mathcal{X} - X_0)}{\vec{T} \times \vec{U} \cdot \vec{S}}$$

But if S, T, U are orthogonal, then $\vec{S} = \vec{T} \times \vec{U}$, therefore the previous equation can be written as:

$$s = \frac{\vec{S} \cdot (\mathcal{X} - X_0)}{\vec{S} \cdot \vec{S}}$$

Solving the above equation:

$$\begin{aligned} s &= \frac{S \cdot (\mathcal{X} - X_0)}{\vec{S} \cdot \vec{S}} \\ &= \frac{\| \vec{S} \| \| \mathcal{X} - X_0 \| \cos\theta}{\| \vec{S} \|^2} \\ &= \frac{\| \mathcal{X} - X_0 \| \cos\theta}{\| \vec{S} \|} \end{aligned}$$

Using Figure B.1 in conjunction with basic trig, it can easily be shown that $\| \mathcal{X} - X_0 \| \cos\theta = \| \vec{s_0} \|$:

$$\begin{aligned} s &= \frac{\| \mathcal{X} - X_0 \| \cos\theta}{\| \vec{S} \|} \\ &= \frac{\| \vec{s_0} \|}{\| \vec{S} \|} \end{aligned}$$

Appendix C

Consistent Edge Generation For Tetrahedra

Assigning edge information to tetrahedra where the edges have already been created by a neighbour implies retrieving the edge reference from the neighbour. There are twelve cases to consider (Figure 4.14), each case dependant upon the prism neighbour's orientation. Note: The following notation is used to to describe each case: $FNX.Y - e_z$ means **F**ace **N**eighbour X tetrahedron Y with edge e_z, where X is one of the three face neighbour prisms, and Y is one of the three tetrahedra that the prism will subdivided into (X,Y $\in \{1,2,3\}$) and edge e_z is one of the size edges in a tetrahedron ($z \in \{1,2,3,4,5,6\}$):

- **Case 1**: Neighbour 1 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_1 is the smallest vertex index in Neighbour 1
  $t_1e_1 = FN1.1 - e_3$, $t_2e_1 = FN1.2 - e_4$, $t_3e_1 = FN1.3 - e_3$, $t_2e_3 = t_1e_1$, $t_3e_5 = t_2e_1$
- **Case 2**: Neighbour 1 has already produced edge information.
  Orientation: $v_2 > v_3$ and v_1 is the smallest vertex index in Neighbour 1
  $t_1e_3 = FN1.1 - e_3$, $t_2e_4 = FN1.2 - e_1$, $t_3e_3 = FN1.3 - e_1$, $t_2e_3 = t_1e_3$, $t_3e_6 = t_2e_4$
- **Case 3**: Neighbour 1 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_1 is *not* the smallest vertex index in Neighbour 1
  $t_1e_1 = FN1.1 - e_2$, $t_2e_1 = FN1.2 - e_2$, $t_3e_1 = FN1.3 - e_2$, $t_2e_3 = t_1e_1$, $t_3e_5 = t_2e_1$
- **Case 4**: Neighbour 1 has already produced edge information.
  Orientation: $v_2 > v_3$ and v_1 is *not* the smallest vertex index in Neighbour 1
  $t_1e_3 = FN1.1 - e_2$, $t_2e_4 = FN1.2 - e_6$, $t_3e_3 = FN1.3 - e_2$, $t_2e_3 = t_1e_3$, $t_3e_6 = t_2e_4$
- **Case 5**: Neighbour 2 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_1 is the smallest vertex index in Neighbour 2

$t_1e_3 = FN2.1 - e_1$, $t_2e_4 = FN2.2 - e_1$, $t_3e_3 = FN2.3 - e_1$, $t_1e_4 = t_2e_4$, $t_3e_6 = t_2e_4$

- **Case 6**: Neighbour 2 has already produced edge information.
  Orientation: $v_2 > v_3$ and v_1 is the smallest vertex index in Neighbour 2
  $t_1e_1 = FN2.1 - e_3$, $t_2e_1 = FN2.2 - e_4$, $t_3e_1 = FN2.3 - e_3$, $t_1e_4 = t_2e_1$, $t_3e_5 = t_2e_1$

- **Case 7**: Neighbour 2 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_1 is *not* the smallest vertex index in Neighbour 2
  $t_1e_3 = FN2.1 - e_2$, $t_2e_4 = FN2.2 - e_6$, $t_3e_3 = FN2.3 - e_2$, $t_1e_4 = t_2e_4$, $t_3e_6 = t_2e_4$

- **Case 8**: Neighbour 2 has already produced edge information.
  Orientation: $v_2 > v_3$ and v_1 is *not* the smallest vertex index in Neighbour 2
  $t_1e_1 = FN2.1 - e_2$, $t_2e_1 = FN2.2 - e_2$, $t_3e_1 = FN2.3 - e_2$, $t_1e_4 = t_2e_1$, $t_3e_5 = t_2e_1$

- **Case 9**: Neighbour 3 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_2 is the smallest vertex index in Neighbour 3
  $t_1e_2 = FN3.1 - e_3$, $t_2e_6 = FN3.2 - e_4$, $t_3e_2 = FN3.2 - e_3$, $t_1e_5 = t_2e_6$, $t_2e_5 = t_3e_2$

- **Case 10**: Neighbour 3 has already produced edge information.
  Orientation: $v_2 < v_3$ and v_2 is *not* the smallest vertex index in Neighbour 3
  $t_1e_2 = FN3.1 - e_2$, $t_2e_6 = FN3.2 - e_2$, $t_3e_2 = FN3.2 - e_2$, $t_1e_5 = t_2e_6$, $t_2e_5 = t_3e_2$

- **Case 11**: Neighbour 3 has already produced edge information.
  Orientation: $v_2 > v_3$ and v_3 is the smallest vertex index in Neighbour 3
  $t_1e_2 = FN3.1 - e_1$, $t_2e_2 = FN3.2 - e_1$, $t_3e_2 = FN3.2 - e_1$, $t_1e_6 = t_2e_2$, $t_2e_5 = t_3e_2$

- **Case 12**: Neighbour 3 has already produced edge information.
  Orientation: $v_2 > v_3$ and v_3 is *not* the smallest vertex index in Neighbour 3
  $t_1e_2 = FN3.1 - e_2$, $t_2e_2 = FN3.2 - e_6$, $t_3e_2 = FN3.2 - e_2$, $t_1e_6 = t_2e_2$, $t_2e_5 = t_3e_2$

Bibliography

[1] David Adalsteinsson and James Sethian. A fast level set method for propagating interfaces, 1995.

[2] Aseem Agarwala. Volumetric surface sculpting. Master's thesis, Massachusetts Institute of Technology, September 1999.

[3] Mark A. Armstrong. *Basic Topology.* Springer-Verlag, 1983.

[4] Fabrice Aubert and Dominique Bechmann. Animation by deformation of space-time objects. *Eurographics*, 16(3), 1997.

[5] Michael F. Barnsley. *Fractals Everywhere.* Morgan Kaufman, 2 edition, 2000.

[6] Dominique Bechman. Space deformation models survey. *Computers and Graphics*, 18(4):571–586, July 1994.

[7] Dominique Bechman and Dominique Gerber. Arbitrary shaped deformations with dogme. *The Visual Computer*, 19(2–3):175–186, 2003.

[8] James R. Bill. Computing sculpting of polygonal models using virutal tools. Master's thesis, University of California, 1994.

[9] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982.

[10] Jules Bloomenthal. *An Introduction To Implicit Surfaces.* Morgan Kaufmann Publishers, 1997.

[11] Paul Borrel and Dominique Bechmann. Deformation of n-dimensional objects. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 351–369. ACM Press, 1991.

[12] Sylvian Brandel, Dominique Bechman, and Yves Bertrand. Stigma: a 4-dimensional modeller for animation. *Eurographics Workshop on Animation and Simulation*, 1998.

[13] Edwin Catmull and James Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[14] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM Press, 1996.

[15] James Davis, Stephen R. Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *First International Symposium on 3D Data Processing, Visualization, and Transmission*, June 2002.

[16] Tony Derose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Computer Graphics (SIGGRAPH)*, pages 85–94, July 1998.

[17] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 19–28. ACM Press/Addison-Wesley Publishing Co., 2000.

[18] Julien Dompierre, Paul Labbe, Mare-Gabrielle Vallet, and Ricardo Camarero. How to subdivide pyramids, prisms and hexahedra into tetrahedra. 8th International Meshing Roundtable, 10-13 October 1999.

[19] Schoenhardt E. Uber die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 1928.

[20] Gerald Farin. *CAGD: Computer Aided Geometric Design.* Academic Press, 1988.

[21] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Virtual sculpture. In *Eurographics*, 1999.

[22] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Practical volumetric sculpting. In *The Visual Computer*, volume 16(8), pages 469–480. Springer, 2000.

[23] James Foley, Andries van Dam, Steven Fiener, and John Hughe. *Computer Graphics: Principles and Practices.* Addison-Wesley, 3 edition, 1993.

[24] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 249–254. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[25] James E. Gain. Virtual sculpting: An investigation of directly manipulated free-form deformation in a virtual environment. Master's thesis, Rhodes University, February 1996.

[26] James E. Gain. *Ehanced Spatial Deformation for Virtual Sculpting.* PhD thesis, Cambridge, 2000.

[27] James E. Gain and Neil A. Dodgson. Adaptive refinement and decimation under free-form deformation. In *Eurographics UK, Cambridge(UK)*, 13-15 April 1999.

[28] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. In *IEE Transactions On Visualisation and Computer Graphics*, volume 7, December 2001.

[29] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 267–274. ACM Press, 1991.

[30] Sarah Gibson and Brian Mirtich. A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electronic Research Laboratory, November 1997.

[31] Sarah F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *IEEE Symposium on Volume Visualization*, pages 23–30. IEEE, ACM SIGGRAPH, 1998.

[32] Igor Guskov and Zoë Wood. Topological noise removal. In *Proceedings of Graphics Interface 2001*, pages 19–26, 2001.

[33] Frank Harary. *Graph Theory.* Perseus Publishing, 1994.

[34] Allen Hatcher. *Algebraic Topology.* Cambridge University Press, 2001.

[35] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. *ACM Computer Graphics*, 26(2):177–184, 1992.

[36] Jing Hua and Hong Qin. Haptic sculpting of volumetric implicit functions. In Bob Werner, editor, *Proceedings of the ninth Pacific Conference on Computer Graphics and Applications (PACIFIC GRAPHICS-01)*, pages 254–264, Los Alamitos, CA, October 16–18 2001. IEEE Computer Society.

[37] Jing Hua and Hong Qin. Haptics-based volumetric modeling using dynamic spline-based implicit functions. In Stephen N. Spencer, editor, *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics (VOLVIS-02)*, pages 55–64, Piscataway, NJ, October 28–29 2002. IEEE.

[38] Arie E. Kaufman. Voxels as a computational representation of geometry. Technical report, University of New York State at Stony Brook, 1994.

[39] Timothy Lambert. An optimal algorithm for realizing a delaunay triangulation. *Information Processing Letters*, 62, 1997.

[40] Jed Lengyel, Mark Reichert, Bruce R. Donald, and Donald P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 327–335. ACM Press, 1990.

[41] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144. ACM Press/Addison-Wesley Publishing Co., 2000.

[42] William Lorensen and Harvey Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California, July 1987).

[43] Marie-Paule, Cani-Gascuel, and Mathieu Desbrun. Animation of deformable models using implicit surfaces. In *IEE Transactions On Vision And Computer Graphics*, March 1997.

[44] Kevin McDonnell and Hong Qin. FEM-based subdivision solids for dynamic and haptic interaction. In David C. Anderson and Kunwoo Lee, editors, *Proceedings of the Sixth Symposium on Solid Modeling and Application (SSMA-01)*, pages 312–313, New York, June 6–8 2001. ACM Press.

[45] Kevin McDonnell and Hong Qin. Dynamic sculpting and animation of free-form subdivision solids. In *The Visual Computer*, volume 18(2), pages 81–96. Springer, 2002.

[46] Tim Milliron, Robert J. Jensen, Ronen Barzel, and Adam Finkelstein. A framework for geometric warps and deformations. *ACM Transactions on Graphics (TOG)*, 21(1):20–51, 2002.

[47] Micahel E. Mortenson. *Geometric Modelling*. Wiley, 1997.

[48] Ken Museth, David E. Breen, Ross T. Whitaker, and Alan H. Barr. Level set surface editing operators. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 330–338. ACM Press, 2002.

[49] Michael J. Muuss and Lee Butler. Combinatorial Solid Geometry, Boundary Representations, and n-Manifold Geometry. In *State of the Art in Computer Graphics: Visualization and Modeling*, pages 185–223. Springer-Verlag, Berlin, Germany, 1991.

[50] Lennes N. Theorems on the simple finite polygons and polyhedron. *American Journal Of Mathematics*, 1911.

[51] Richard Parent. A system for sculpting 3-d data. *SIGGRAPH*, 11(2):138–147, 1977.

[52] Ronald N. Perry and Sarah F. Frisken. Kizamu: A system for sculpting digital characters. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 47–56. ACM Press / ACM SIGGRAPH, 2001.

[53] Mark Rioux. Laser range finding based on synchronised scanners. *Applied Optics*, 23(21):3837–3844, 1984.

[54] William Schroeder, William Lorensen, and Scott Linthicum. Implicit modeling of swept surfaces and volumes. In R. Daniel Bergeron and Arie E. Kaufman, editors, *Proceedings of the Conference on Visualization*, pages 40–45, Los Alamitos, CA, USA, October 1994. IEEE Computer Society Press.

[55] Thomas W. Sedarberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH*, pages 151–160, 1986.

[56] Nilo Stolte and Arie Kaufman. Discrete implicit surface models using interval arithmetic. In *Second CSG Workshop on Computational Geometry*, October 1997.

[57] William P. Thurston and Silvio Levy. *Three-Dimensional Geometry and Topology.* Princeton University Press, 1997.

[58] Mark Tigges, M.S.T. Carpendale, and Brian Wyvill. Generalized distance metrics in implicit surface modelling.

[59] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 151–156. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[60] Joe Warren. Subdivision methods for geometric design. 1995.

[61] David Wells and John Sharp. *The Pengiun Dictionary Of Curious And Interesting Geometry.* Pengiun, March 1992.

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