3D Visualisation of the Laetoli Footprints on the Internet

Simon Taylor

University of Cape Town

February 2000

Abstract

This thesis examines the use of 3D visualisation over the Internet as a tool for analysis and presentation of the Laetoli footprints. It describes the data used in the project and the methods used for checking and processing the data. Principles for creating 3D models are explained in detail. The tools used in the project are discussed, particularly VRML-based tools and how these tools integrate with the Internet. The discussion includes the design of the tools, which have been written for manipulating a VRML scene and for the creation of user-based interaction. The thesis also outlines some of the benefits and limitations of 3D visualisation as an analysis tool and it's suitability for use over the Internet. Finally, hardware and software issues pertaining to 3D visualisation over the Internet are discussed in terms of current status and future possibilities.

Table of Contents

List of Figures

List of Tables

Glossary of Terms

3D Visualisation

This essentially involves describing objects in a way that depicts 3 dimensions. This is best described by comparing 2D and 3D visualisation. A plain image on a computer screen is a 2D image, where every pixel has a fixed, unchanging value. A 3D scene, however, can be viewed from an infinite number of viewpoints. 3D scenes can be enriched with sounds and other forms of media such as video.

Application Programming Interface (API)

A means of communicating between the user and the operating system. A graphical user interface (GUI) is used as a means of translating the intentions of a user into a command that is understood and carried out by the operating system.

Behaviour

In terms of 3D visualisation, behaviour is what makes the scene interactive. Behaviour of the scene includes all operations that change the state of objects that make up the scene.

DEM (Digital Elevation Model)

A DEM is defined as a "digital cartographic representation" of the elevation of land at regularly spaced intervals in *x* and *y*.

Ortho-image

An ortho-image is an image showing objects in their true orthographic (*x*, *y*) positions and is therefore geometrically equivalent to a map. This means that it can be used for direct angular, distance, area and position measurements without making corrections for image displacements (within the limitations of the projection used). The principal difference between an ortho-image and a map is that the former uses actual images of objects or features, while the latter utilises lines and symbols to represent objects and features.

Plugin

A software module that adds a specific feature or service to a larger system. For example, there are number of plug-ins for the Netscape Navigator browser that enable it to display different types of audio or video messages (PC Webopedia, 1999).

Render

The display of 3D pixel information on a computer screen is known as rendering. It is the process of adding realism to computer graphics by adding 3D qualities such as shadows and variations in colour and shade (Rozendaal, 2000).

A scene (also referred to as a universe)

A scene includes all objects, background colours, lighting, sounds etc. which make up a 3D model.

Static scenes vs. Interactive scenes

A static scene is one in which the elements of the scene cannot be changed by the user. The users position relative to the scene changes enabling 3D viewing to take place, but no mechanism is available for altering the elements of the scene. Interactive scenes allow elements of the scene, such as colours, to be altered by the use of some type of interface, typically buttons, checkboxes and so on.

Virtual Reality

Virtual reality implies a true 3D experience using glasses or headgear.

Chapter 1 Introduction

1.1 3D Visualisation

In the last 30 years, the way that data is stored and presented has been radically transformed. Computer technology has created opportunities to present results to more people, more economically and in a more descriptive fashion. Data stored in digital format has benefits over traditional data storage and the benefits continue to increase with each advance in the field of computers. 3D visualisation is such an advance which introduces some unique benefits.

3D visualisation is used to depict an object in three dimensions using computer graphics which enables one to view the object from any desired perspective. One has the ability to control the way the data is viewed unlike other forms of presentation where the result (be it in the form of text, images video clips etc.) lacks interactivity and therefore cannot be explored with the same effect. 3D visualisation has been used successfully in various fields such as engineering, medicine and architecture. This thesis deals with archaeological data, and the following benefits are suggested for the use of 3D visualisation in the field of archaeology:

- *Researchers can share information easily.* Previously, information may have been shared by means of photographs, sketches and text, but this technology enables a more descriptive 3D representation to be presented.
- *Information can be easily updated.* As new information becomes available it can be added to the 3D model without interfering with existing data. Field sessions may acquire data over a long period of time and by using 3D visualisation, data can be altered or supplemented easily.
- *Information is more readily available to the public.* So called "virtual museums" are becoming more common where a user may walk through a building and sometimes interact with the items on show. This is not to say the tour is truly authentic, but it does have advantages in that the items do not have to be looked after in a museum where there is a risk of damage. A more significant advantage is that such displays are available to a wider audience, including many people in remote areas. Such Virtual tours can be useful as an educational tool by showing historical sites and items. Sounds and text can be added which tell important facts making it a useful learning tool. The 3D models are becoming more realistic as technology and skills develop. At present, models are already available which are indistinguishable from high quality photographs.
- *Extra information may be provided.* The computer is able to perform operations on a model that cannot be performed otherwise. Models can be stretched, colours adjusted, lighting altered, texture overlaid, and an endless list of other possibilities which may present

information previously unavailable. Simulations can also be performed which also provide unique benefits for testing hypothesis.

- *Information is preserved.* Due to the fragile nature of archaeological sites, tampering or interference must be avoided. An accurate 3D model not only enables work to be done without the fear of damage, but it also serves as a record in case the original site is damaged or destroyed. Computer generated models can be used as an accurate and reliable guide for future restoration and renovation.

The Internet has the most substantial impact when it comes to presenting results because a worldwide audience can be reached with a minimum expense. This project attempts to make use of both the Internet and 3D visualisation and in doing so, present the data in a meaningful and interactive way to the widest audience.

1.2 History

Mary Leakey's interest in fossils found in Laetoli led her to revisit the area in 1935. Volcanic deposits had preserved material over a million years old which was thought to contain fossils and perhaps tools. In 1977, a site exposed by erosion was found to contain a hominid footprint that has since been dated as being 3.6 million years old – the earliest evidence of hominid upright walk. The footprints were almost indistinguishable from a modern human footprint. The excavated area was documented and reburied for preservation.

This discovery has fundamental importance for the understanding of human origin. Not only does it assist our knowledge of the time frame of evolution, but it also provides information that cannot be obtained from fossilised bones. The footprints give an indication of the tissue making up the foot and the gait of the early hominids.

Almost 20 years after the original field work was done, it was decided that the site should be re-excavated, documented and preserved using modern, more accurate techniques. A team comprising members of the University of Cape Town were involved in two further field campaigns in 1995 and 1996. Photogrammetric techniques were used to accurately record spatial information of the footprints and the trackway.

Archaeological discovery is incomplete without recording, but in most cases documentation leaves frustratingly inadequate accounts – sometimes of sites which have since been destroyed (Greene, 1983). Data gathered from the Laetoli site has formed the basis for this project along with others also concerned with different aspects of the Laetoli footprints.

1.3 Objectives

This thesis is concerned with the distribution of information pertaining to the Laetoli footprints via the Internet, with the focus on an interactive 3D application. The objectives are summarised as follows:

Primary Objective - To create a visualisation tool that can be accessed over the Internet and used for analysis,

Secondary objective - To create a Web page containing static data.

These objectives will be made clear in chapter 3 (Project Framework) which gives a more detailed account of the objectives, the tools required and views of some of the project expectations.

The thesis also attempts to investigate 3D visualisation in an effort to answer the following questions.

- To what extent is 3D visualisation restricted by using conventional hardware?
- Does 3D visualisation provide a practical means for analysing data?
- If so, can it be suitably used as an Internet application?

1.4 Thesis Outline

Chapter 2 (Literature Review) introduces some basic concepts of 3D visualisation and takes a look at some similar work done in other archaeological applications.

Chapter 3 shows the layout of the project and the tools needed for the project implementation. It gives a more detailed account of the project objectives and expectations.

Chapter 4 explains the theory of VRML – the program used to create the 3D models. It explains the development of the VRML standard, the way it fits into the infrastructure of the Internet, it's basic structure and a look at the possibilities of VRML development in the near future.

Chapter 5 deals with the processing of primary data. The methods to ensure the data is free of blunders and the programmes written for this purpose are discussed.

Chapter 6 gives a detailed account of the design and results of the work done in 3D visualisation. It describes how the models were created and the methods used for refining their appearance. It also deals with the tools that were programmed for scene manipulation and analysis.

Chapter 7 concludes the thesis and describes some shortcomings of the project. It attempts to answer the questions posed above, regarding certain aspects of 3D visualisation.

Chapter 2 Literature Review

2.1 Introduction

This chapter discusses some examples of work done in the field of archaeology using 3D visualisation. Before they are discussed, some general concepts and understanding of 3D visualisation is required. As a means of clarifying what 3D visualisation is and what some of its benefits are, it is compared to 2D visualisation (i.e. viewing a picture). Although there are many differences, four are mentioned which best show the benefits of 3D visualisation:

- *A 3D scene can be viewed from an infinite number of positions.* The nature of a 2D image results in certain restrictions. Only one side of the object can be seen, objects can be obscured, and if objects are distant and cannot be seen properly, nothing can be done. In 3D visualisation none of these problems exist because the user is able to move relative to the scene. The ability to rotate the scene provides other advantages such as creating a better impression of shape. 2D images do not always convey shape well. Much of our perception of shape comes from viewing an object from different angles which can a only be done in 3D visualisation.
- *There is control over the objects making up a 3D scene.* A mouse click on an object within a 3D scene can be recorded as such, and a change to the scene can be made. Having control over the objects means that it's properties can be changed as desired (e.g. an object could be made semi-transparent in order to reveal what lies behind it). This functionality cannot be performed on 2D images.
- *The objects of a 3D scene have dimension.* Each object must have a dimension because this information is required when designing the model. This information can then be retrieved when working with the result. This is one of the main contributing factors making 3D visualisation such a powerful tool for analysis.
- *The viewer can be incorporated into a scene.* A viewer can be represented as part of a scene (an avatar) which a third party observer could view. The avatar moves according to user actions.

The literature review uses four brief case studies to show some of the work done using 3D visualisation as a tool benefiting archaeology. They give an indication of it's versatility ranging from use as a tool for analysis, to being used for tourism (Stonehenge).

2.2.1 The Temple of Sulis Minerva (Woodwark, 1991)

By today's standards this project may not seem impressive, but it is significant as it is one of the earliest cases of the use of computer graphics for archaeological reconstruction. The project began in late 1983 when a model of the Temple of Sulis Minerva and nearby areas were reconstructed in

3D. The construction was made using only one primitive solid – the planar half space. Approximately 2300 of these planes were used in the model and the result was a number of birds eye views of the site (no animation or user interaction was available). Some of the views were used as images on postcards to describe the area.

A year later, construction of an interactive 3D model of the Roman Fortress Baths began. The result was a more detailed model comprising over 7000 planes as well as features for walkthroughs. Although there was no similarity in the methods of data collection to today's digital photogrammetry techniques, the principle of using computer graphics for archaeological reconstruction was similar. These 3D models were used to describe the layout of the Caerleon Baths to visitors.

Both these models were designed to show the layout of the buildings to visitors rather than depicting highly detailed and accurate models. Advanced data collection methods were not used and photographs were not overlaid on the model. The project was important in that it was possibly the first demonstration of 3D technology in the field of archaeology.

2.2.2 Stonehenge (Sims, 1997)

The 3D model of Stonehenge is one of the most publicised and well known examples of the use of computer graphics to represent archaeological sites. The Stonehenge model is accurate to 1cm and as well as representing the monument and nearby ground, it also accurately represents the positions of the stars in the sky at any given day of the year. This model also lets users trace the development of the monument over the last 10 000 years. The 3D model was presented in 1996 and showed the layout of Stonehenge today as well as what it might have looked like without any made objects nearby (e.g. roads that pass nearby). A less detailed model was produced using VRML and was made available for access via the Internet. The main purpose of the model is to give tourists a better understanding of the monument. It also serves the purpose of describing Stonehenge to those who are unable to visit the site.

2.2.3 The Visir Tomb (Palamidese *et al*, 1993)

This case is concerned with the virtual reconstruction of an Egyptian tomb, the largest tomb of Memphis. Several field campaigns to the site revealed rich architectural and cultural finds. Of particular interest was the funerary equipment of prince Bakenrenef, visir (advisor) of the Pharaoh and his descendants who were buried in an underground network of shafts and rooms.

The site remained in good condition until interest in Egyptian antiquities grew. Since then, many blocks making up the tomb were stolen and sold. Many of these blocks have since been found and restoration began in 1985. Restoration consisted of:

- Repositioning/repairing damaged blocks,
- Restoring decorations,
- Consolidation of existing paintings and sculptures.

During the restoration process, a large amount of geological, architectural and artistic data was gathered which would later be used in visualisation techniques.

The use of visualisation in this project began in 1985 when information of 2000 of 8000 of the blocks (including images of the paintings on each block) was compiled for the first hypothesis for reconstruction. This proved to be a very useful tool and it was decided that these essentially 2D modelling techniques would be extended to 3D modelling to assist in the following:

- Testing restoration hypothesis,
- Creating a complete model. Many existing blocks are still located in museums and others are likely never to be recovered. A complete physical restoration is thus not possible and a computer model is the most feasible way of creating a complete restoration.

The visualisation model was divided into two categories, one dealing with volumetric data, the other decorations and wall materials.

The results achieved from the 3D visualisation techniques allow the blocks to be positioned in order to test hypothesis. Those blocks that still remain in their correct positions or those where the hypothesis has proven successful can be separated from the remainder by colour so as to avoid confusion. Other blocks can be separated into categories such as "blocks found in museums" and so on. Another feature is a walkthrough of the tomb where a user is taken on a guided tour through passages and walkways, turning their attention to paintings or statues where necessary.

The model has proved successful particularly in the field for which it was designed – restoration. The drawback has been that highly sophisticated hardware is required as well as programming knowledge to manipulate the scene. This is likely to change with the focus on a more user friendly environment for use in the field of virtual tourism.

2.2.4 The Florentine Pieta (Abouaf, 1999)

The Florentine Pieta is a statue carved by Michelangelo. It depicts the figure of Christ being lowered from the cross into the virgin Mary's lap, Joseph looking on and Christ touching Mary Magdalene. Two years after Michelangelo stopped working on the unfinished piece, he attacked the statue with a sledgehammer. A second artist repaired the damage and completed the unfinished sections.

By using modern techniques of digital photogrammetry and 3D visualisation, a team funded by IBM planned to investigate various aspects pertaining to the statue. The main focus was to determine who was responsible for sculpting each part of the statue. Various other areas of research were also under investigation including historical, aesthetic and scientific aspects.

The proportions of the figure in the statue seemed peculiar – some elongated and others very small. It was suspected that the unusual proportions were because the sculpture was meant to be viewed from below and should be elevated to appreciate the correct effect. Researchers were not able to move the work due to its fragile nature and of course disassembling the work was not permitted. A 3D model that could be freely moved and manipulated would become a very useful tool for researchers. Research began in the mid nineties and is expected to continue into the year 2000. Some of the early results have been impressive with an available measuring accuracy from on the statue of approximately 1mm. Current and future advantages suggest the following benefits:

- *Measurement.* The accurate 3D model allows relevant dimensions and proportions to be calculated,
- *Convenience.* Various people can work on the model at the same time and research is not restricted to times when the statue is made available. Conditions at the site are not ideal for research and without the 3D model, lighting etc. cannot be altered,
- *Problem Solving.* Raising the statue to any position in a virtual environment gives the perspective the artist had when carving or the perspective that the statue was intended to be viewed from,
- *Documentation.* An accurate 3D model serves as useful educational data and a record for future conservation purposes.

Chapter 3 Project Framework

3.1 Introduction

This thesis is based upon the need to effectively display spatial and attribute data pertaining to the Laetoli footprints. A Web page was designed to serve this task. This chapter briefly illustrates the layout of the Web page and introduces the tools necessary for its design and implementation.

3.2 The Web page

The overgrowing popularity of the Internet has made it an invaluable source of information. Almost every conceivable area of research has presented information on the Internet in a very short space of time since it's inception. Not only does it contain huge resources of information, but since all data is in digital format the data can be easily sorted, copied or manipulated. Hyperlinks make data easy to locate, 3D displays allow a better visual perception of an object, and various computer tools provide functionality which makes digital data an effective way to convey information.

It is not practical to include all types of data into a Web page though, so before beginning with the implementation of a project such as this, the suitability of the Internet for the task at hand needed to be established. The Web pages that were to be represented are categorised as follows:

"Static"

This type of Web page displays information which cannot be modified by the user and where a set layout is expected to fulfil most needs. The majority of Web pages can be classed in this category where the information consists of text and images and forms part of a static display. The user may view (or listen to) the information but cannot interact with it in any way. A large amount of information pertaining to the footprints and the Laetoli site was available in the form of text and images which could be displayed using static Web pages (secondary objective).

"Live"

In some cases it may be required that users interact with the data to suit their own needs. The most important part of the thesis was to create visualisation tools that could be accessed over the Internet (primary objective). A visualisation tool is classed as live because the user's actions must be interpreted by a program which modifies the information accordingly. Research was carried out to investigate the tools available to perform such a task, and it was found that 3D visualisation on the Internet was a viable option.

Much of the remainder of this chapter is concerned with the tools used for both the static and live part of the web page. Two phases of development of the Web page would be required; one component aimed at providing information of interest to the general public and another component providing more in-depth information and 3D functionality aimed at the user with a more specialized involvement in the project. The Web page contains all the results of the project and show its basic layout.

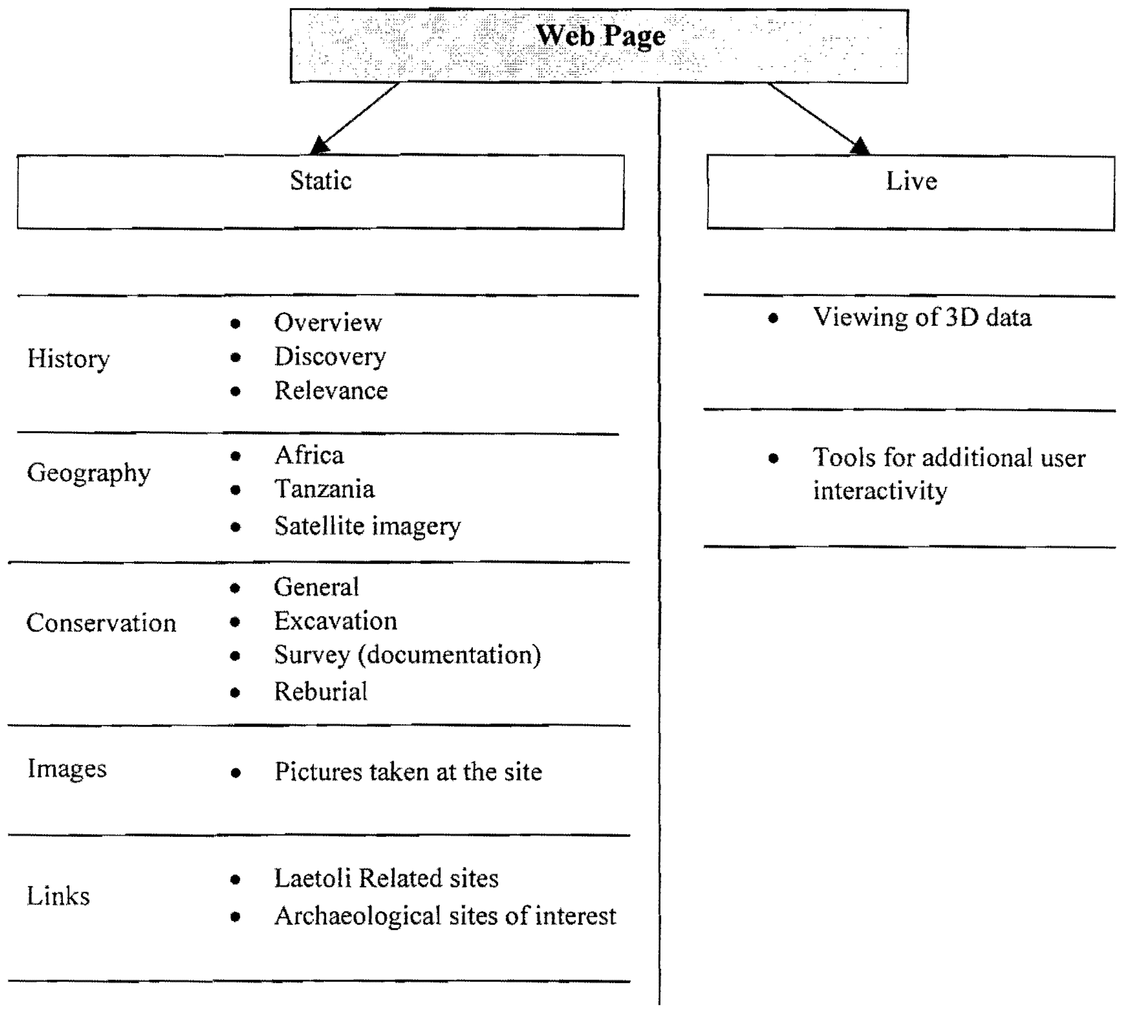


Figure 3.1: Project Structure

3.3 Tools required for the Creation of the Web page

3.3.1 The Static Side

HTML editors (e.g. Microsoft FrontPage, Netscape Composer) have made compiling a web page as simple as writing a text document and there is no longer a need to learn HTML code in order to produce an effective Web page. Following some basic guidelines which make the data easily accessible and readable is the most important tasks. The most popular pages are not necessarily ones with the best graphics or newest animations, but rather the ones with content of interest. The intention for the static Web pages was simply to provide the information that the user requires and to make the layout clear and simple to allow the required information to easily located.

3.3.2 The Live side

At one time, any interaction with the data comprising a Web page was impossible. The code used for creating Web pages (HTML) includes text and refers to images which are interpreted by the browser (e.g. Netscape). If a user's input is to be processed and used to interact with the data, a programming language is required and since HTML is not a programming language, an alternative is needed. The Java programming language was used because it can be embedded in an HTML document. (Although Java forms the basis for any interaction on a Web page, another type of software was also needed to deal specifically with the visualisation side of the project. This is VRML and is dealt with in chapter 4)

3.4 Java

3.4.1 Introduction

What is it that makes Java so special? A simple answer is that Java is platform independent. This means that Java can (theoretically) run on all types of computers, gadgets and other devices (such as phones). Having a language that is understood by all devices makes communication between them easier than ever before. What follows is a brief look at the history, characteristics and future of Java.

3.4.2 History in brief

Java is a relatively new product having been released less than five years ago. This is a summarised list of the important dates in Java's history (Naughton, 1999).

Table 3.1: Important Dates in Java's Development

1991, June	Gosling starts working on the "Oak" interpreter, which, several years later (following a trademark search), is renamed "Java"
1995, May 23	Sun formally announces Java and Hot Java at SunWorld '95
1995, May 23	Netscape announces its intention to license Java for use in Netscape browser
1995, Dec 4	Sun and Netscape announce JavaScript, a scripting language based on the Java language which is designed to be accessible to non-programmers
1995, Dec 4	Sun, Netscape and Silicon Graphics announce new software alliance to develop Internet interactivity tools
1995, Dec 6	IBM and Adobe announce licensing agreement with Sun for use of Java
1995, Dec 7	Microsoft announces plans to license Java

3.4.3 General Characteristics

Having mentioned that the main attraction of Java is it's platform independence, this section will describe how this is achieved and what makes this so significant. It will also touch on some of the other characteristic features which make up Java so as to give an impression of it's makeup and of what type of application it is best suited to.

Platform Independent

There are a number of benefits associated with a program that can be used on all operating systems. Having a means to communicate between different devices means that software written on one system (e.g. Windows) can be used on any other system (e.g. Sun's Solaris). This is the obvious advantage which has great importance when it comes to networking and the Internet in general. Networking computers may require communication between different systems, and using a language understood by all types of systems simplifies matters. The Internet is also greatly affected by Java because a program can be embedded in a web page and can be interpreted by all Internet users, whereas any previous method would require different programs to be written for each operating system.

Java's platform independence is made possible by using a so called Java Virtual Machine (JVM) (the Virtual Machine is not unique to Java). This is a translator that turns Java code into tailored instructions for the particular computer (or device) it is working on. Java code is distributed in byte-code format which means it will not differ on different machines since all operating systems read byte-code. The reason other programs are not able to be distributed across all platforms is because the code is compiled for a particular system which makes use of the methods or libraries available to that system only. This means that most programs will be interpreted differently on various operating systems because certain essential classes and libraries used on the original machine when the program was written and compiled may not be available on a machine using a different operating system. In simple terms, there is not a standard method for interpreting compiled code written in most languages, Java being an exception. A standard method for interpreting code is available to Java (Java's virtual machine) because it functions independently and does not rely on the system libraries. Since Java is potentially O/S independent, it is a threat to any proprietary system (e.g. Windows) (Newmarch, 1997).

The Language

The logic used for different programming languages is much the same, it is a matter of becoming familiar with the syntax. Java shares much of its basic syntax with C++ and is generally considered easy to learn.

Java is object oriented which requires it to have the following characteristics (Rumbaugh *et al*, 1991).

Identity - data is grouped into entities called objects,

Classification - objects with the same data structure and behaviour are classified into classes,

Polymorphism - the same operation may behave differently on different classes,

Inheritance - the sharing of attributes and operations among classes based on a hierarchical relationship.

Java programmes are classed as either applets (for Web pages) or applications (normal standalone software). Applets are unique to Java and are programmes that are used on the Internet by being embedded in HTML code. There unique ability is derived from their platform independence so that they can be downloaded with a Web page and work on any system. An applet could be as simple as a button that opens a new Web page, or could resemble a fully fledged program performing advanced tasks. Applets have transformed the way the Internet is used by expanding its capabilities to include functionality one may expect from software running locally.

3.4.4 Strengths & Weaknesses

Strengths

Apart from the points mentioned above these are the main strengths of Java:

- *Security.* Java applets cannot access information on the local disk or query its contents in any way. There is not the security risk present with scripting languages for example, which can read certain information. Java also cannot run local programs on its behalf (Newmarch, 1997). A more detailed discussion of security maters is beyond the scope of this thesis. The two points mentioned above are the most significant of the security issues which alone make it more secure than any previous options.
- *It is freely available.* The Java Development Kit and comprehensive documentation is freely available for download.

Weaknesses

- *Speed.* This is really the main concern with Java. The chief contributing factor is the need for the JVM which adds a layer of interpretation. The JVM is required to interpret byte code which is a step not performed by other programming languages. Any layer of interpretation adds an order of magnitude to the speed of a program which accounts for much of the loss of performance of Java over other programming languages. There are some solutions to this problem such as a method called Just in Time (JIT) compilation. This compiles native code just before execution which requires the JVM to be a compiler as well as an interpreter. Speed considerations are normally a less crucial factor for applets (as opposed to applications) because most applets are very small in size and speed factors are not as noticeable.

3.5 Project Aims

This chapter has outlined the tasks to be performed and described the tools needed for their implementation. The two objectives mentioned were:

- To create a visualisation tool that can be accessed over the Internet and used for analysis,
- To create a Web page containing static data.

These are general statements that say little about the project expectations. A realistic expectation can only be made when a certain knowledge of the project and the capabilities of the tolls are understood and it was not until at least three months had passed that some firm idea of project aims were developed. In this time, understanding of the following was developed:

- History of the project and the work done up to that point,

- Web page design,
- 3D visualisation principles,
- Visualisation programming (capabilities and restrictions).

It was then that the project aims were made which altered little throughout the thesis.

Learning about the history and importance of the Laetoli footprints developed an interest in telling people about the project. In various stages of development, groups and individuals were given a run down of the project and shown the work done up to that point. The reception was always one of interest whether the audience was made of school children who had never heard of the Laetoli footprints, or archaeology lecturers who knew every detail of the discovery.

The interest for the specialist and the layman is obtained for different reasons. It is expected that the interest of the layman will be captured from the description of the historical angle without providing in-depth information and by appreciating the graphic representation. The interest of the specialist (someone working in a related field such as archaeology) is expected to be obtained by demonstrating the benefits of such technology (visualisation and the Internet) for their possible use. Supplying more detailed information in the Web page may be of interest, but the focus was more on creating visualisation tools and demonstrating how they can assist those working with the footprints. If someone working on the footprints finds the result interesting it is likely to mean that the result is of some benefit to them. It is hoped that the principles can be seen to be applied to other projects whether it be merely for displaying an object in 3D or in the creation of a similar specialised program assisting analysis. The aim of the project was to create this interest for all users and of course to learn about 3D visualisation in the process.

Chapter 4 VRML

4.1 Introduction

VRML stands for Virtual Reality Modelling Language and is the standard language for 3D visualisation on the Internet. The purpose of this chapter is to introduce the basic concepts of 3D visualisation programming and the way in which VRML in particular is designed. An attempt has been made to keep the technical details to a minimum, however certain technical knowledge is required if there is to be an understanding of the capabilities and limitations of 3D visualisation.

4.2 History of VRML

In 1992, a C++ toolkit called "Iris Inventor 3D toolkit" was released. This was a Silicon Graphics project designed for the creation of interactive 3D applications. In 1994, the second major revision of Inventor was released. It was named "Open Inventor" because it was portable to a number of platforms and it was based on "OpenGL". The Open Inventor format was used as the basis for the design of VRML 1.0. (Carey & Bell, 1997).

In 1994, a call for proposals for 3D specification on the Internet was released. The suitability of Open Inventor was seen and the proposal was put forward. This was accepted and the first draft of VRML 1.0 came about. In 1994, at the Second International conference on the World Wide Web, the specification was published (Carey & Bell, 1997). VRML was intended to become the standard for Internet based visualisation, and has since been adopted as such. This is not to say it is the most powerful visualisation tool, but rather that it is a standard that runs on a wide variety of computers and provides most of the common 3D functionality. It was not intended to replace specialised visualisation tools designed for specific platforms.

By the end of 1995 it was decided that VRML 1.0 was missing key features (such as animations and user interaction). In August 1996, VRML 2.0 was released which addressed these problems. In 1997 VRML was established as an Internet standard by the ISO (Landes, 1998). For the remainder of this thesis, when speaking of "VRML", VRML 2.0 is implied.

4.3 VRML and the Internet

A VRML document is platform independent and takes the form of an ASCII file describing a 3 dimensional scene. VRML runs within a web browser (this example refers to Netscape) as opposed to most other visualisation packages such as Java 3D which are standalone applications. The VRML console provides the basic movement tools used to navigate through the scene and a part of the plugin which explains the different look and feel of the same VRML file using different plugins. An ASCII text file describing the objects in the scene is interpreted by Netscape which then loads the appropriate software needed to render the scene.

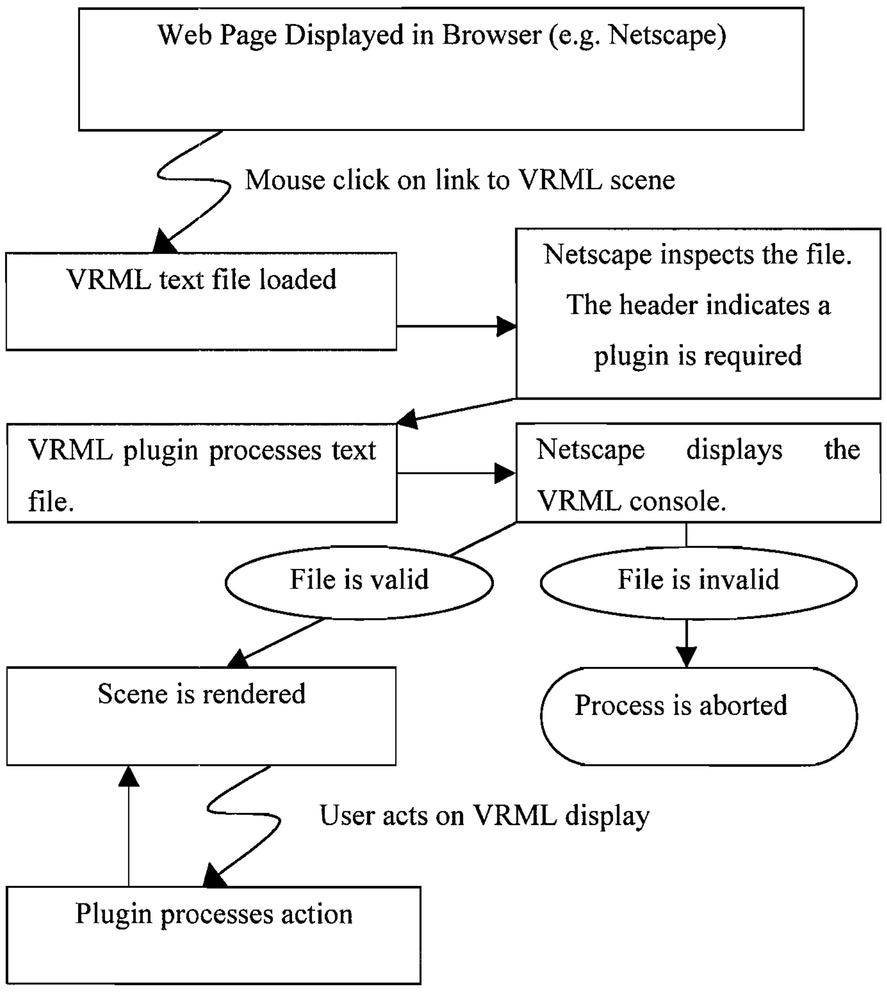

Web Page Displayed in Browser (e.g. Netscape)
Mouse click on link to VRML scene
VRML text file loaded
Netscape inspects the file. The header indicates a plugin is required
VRML plugin processes text file.
Netscape displays the VRML console.
File is valid
File is invalid
Scene is rendered
Process is aborted
User acts on VRML display
Plugin processes action

Figure 4.1: The Relationship Between VRML and the Browser

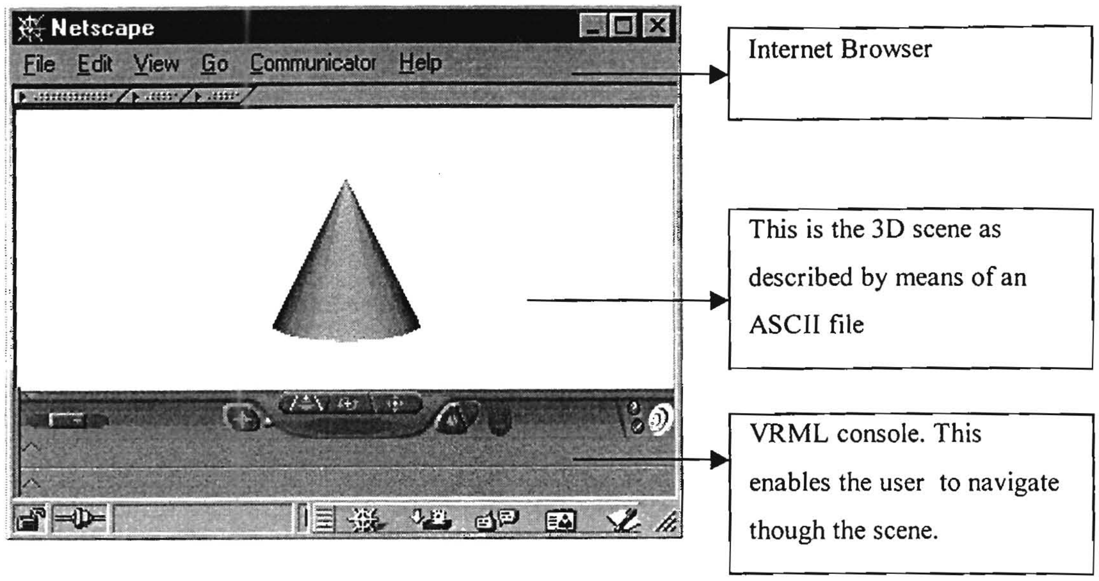

Netscape
File Edit View Go Communicator Help
Internet Browser
This is the 3D scene as described by means of an ASCII file
VRML console. This enables the user to navigate though the scene.

Figure 4.2: A Basic VRML Scene

VRML is designed to make use of existing standards used by the Internet. It is possible that some of these standards may have shortcomings when combined with VRML. Using existing standards make VRML development easier because there is not the difficulty of attempting to establish non-standard or unrecognised formats to the Internet. (Carey & Bell, 1997). Some of the formats which have become a standard for Internet use include:

- .JPG/.GIF image formats,
- .WAV sound format,
- JavaScript scripting language.

Fitting VRML into the existing infrastructure of the Internet lead to a number of constraints which are described in the following chapter.

4.3.1 Design Constraints

There are a number of different 3D languages and each has it's own advantages and disadvantages which are largely dependant on the purpose the language is designed to serve. VRML was designed for the purpose of 3D visualisation over the Internet and as a result, has certain inherent design restrictions in order to make it practical for such a task:

- *File size.* Reduced file size decreases download time needed to transfer files across the internet. Files must therefore be efficient at describing the scene using a small amount of code (e.g. by avoiding unnecessary duplication of data).
- *Portability.* If VRML was to become the standard of 3D visualisation on the Internet, it was important that VRML files could be viewed on a large range of hardware configurations.

- *Easy to configure.* If VRML was to become widely accepted and used, it was to be made easy to set up even for inexperienced users. Setting up software to work on the Internet sometimes involves downloading a plugin that interprets the code and runs it on the browser. The plugin can contain functions and modules used to perform tasks that are not standard operations performed by the browser. Plugins are inconvenient and sometimes difficult to install especially for the inexperienced user and it is important that this inconvenience be kept to a minimum. VRML achieves this well because modern browsers (e.g. Netscape Communicator 4.5) do not require a plugin to view VRML files. Older browsers do require a plugin though which is typically about 3Mb in size.
- *Sharing work.* One of the key considerations for making a 3D visualisation tool successful on the Internet was to be able to use other models found elsewhere on the internet and include them as parts of ones own scenes. The ability to link files and use URL's of other work in ones own work is a great asset of VRML.

4.3.2 Syntax Restraints

The syntax of VRML and most 3D visualisation programs follow a similar hierarchical structure which will be explained later. VRML does however have some syntax characteristics that differ from the norm which arise from demands created when designing for use on the Internet.
VRML file format was designed so that it could become a good file interchange format (FIF) - a format which is easy to read and to write. As an example, HTML is a particularly good FIF which is a major contributing factor to its success (Carey & Bell, 1997). A good FIF encourages the development of editors to read and write new files. This applies to the VRML file format files and enables programs to be created which convert the data available in existing forms (e.g. contours in DXF format) to VRML file format with the minimum of difficulty. In contrast to this, programming languages make for very poor FIF's because of their less predictable structure. Although VRML is normally referred to as a programming language (and is referred to one throughout this thesis) it is not entirely accurate because it is essentially a scene description format.

Sections of code must be portable. It must be easy to copy parts of code from one file and add it to another with the minimum of changes. Objects making up a scene are clearly separated in the VRML file. It is possible to identify and extract a single object of interest from a complex file. It is even possible to extract any lighting operations performed on the object of interest. This is an important contributing factor that enables work to be shared.

4.3.3 Hardware Constraints

It is important to cater for all possible users of the Internet, which means allowing for a wide variety of hardware configurations. VRML is platform independent which allows users of all operating systems to access VRML files but the remaining issue is that of hardware performance. Users on low end hardware must be accommodated as much as reasonably possible which imposes a constraint on VRML. There are two general categories of hardware restrictions to be considered:

- *Internet Connection.* On faster more reliable Internet connections, it is possible for VRML to handle very large scenes distributed via the Internet. The Internet is not always reliable though, and if one or more of the scenes does not load (e.g. a broken link), VRML is still able to continue with the remaining files.
- *Conventional Hardware.* VRML works very well even on machines with relatively poor hardware. It is difficult to define what poor hardware is because this changes over time, and there is no boundary which separates good and poor hardware. Hardware requirements also vary from task to task and visualisation is likely to make a higher demand on hardware than more common tasks such as word processing. Hardware is discussed in more detail later in this chapter. The point is that the VRML browser is able to cater for a very wide range of configurations by scaling the complexity of the model in order to trade off image quality with performance when necessary.

4.3.4 Summary

The constraints mentioned above lead to VRML being limited in certain aspects, and very strong in others. The results are summarised as follows.

Strengths

Almost all aspects of VRML are tailored towards making it integrate well into the Internet which is leading to the rapid growth of VRML recourses. Sharing work done in VRML is a trivial task since results are distributed via the Internet, whereas other 3D packages may not have the same growth because work is more isolated.

Weaknesses

The limitation on file size and the need for small plugins restrict the ability to have rich features that one would expect from a fully fledged 3D package (which can include a large number of modules or functions). It is difficult to perform specialised tasks and to create user interaction in VRML because many of these modules that are standard in some packages have to be programmed without the help of any basic mathematical libraries.

These trends are proven by the 3D files available on the Internet. The ease of sharing VRML files on the Internet is shown by the vast number of VRML scenes as opposed to other 3D packages. The difficulty in creating tools for user interaction is also confirmed by the lack of interactivity found in most of these VRML files. Comparing this to Java3D for example, where the opposite trend is shown, i.e. not as many resources on the Internet, but a large percentage of those available have rich user interaction capabilities.

4.4 VRML – Elementary Structure

Three-dimensional dynamic systems are typically concerned with the animation of models arranged in a hierarchical fashion. Such systems usually need to contain the following information in the graphical data structure: (Thalmann & Thalmann, 1991)

- The shapes of the models, described in a local reference frame;
- Their position, orientation and scale in Cartesian space;
- The hierarchical relationship between the different reference frames (nodes);
- The rendering attributes of the different models.

This kind of knowledge can be encapsulated in an object-oriented structure, with the responsibility of handling the different types of information decentralised among specialised classes. This is the case in VRML, and these topics will now be introduced.

4.4.1 The coordinate System

By default, the coordinate system for VRML is right-handed, with the positive *X*-axis to the right, the *Y*-axis up and the *Z*-axis out the screen. When looking at a 3D scene, a viewpoint has been specified that indicates the initial orientation and position of the viewer, relative to the scene. When navigating through the scene, either the scene moves, or the translation and orientation of the viewpoint changes in order to create the appearance of a moving scene.

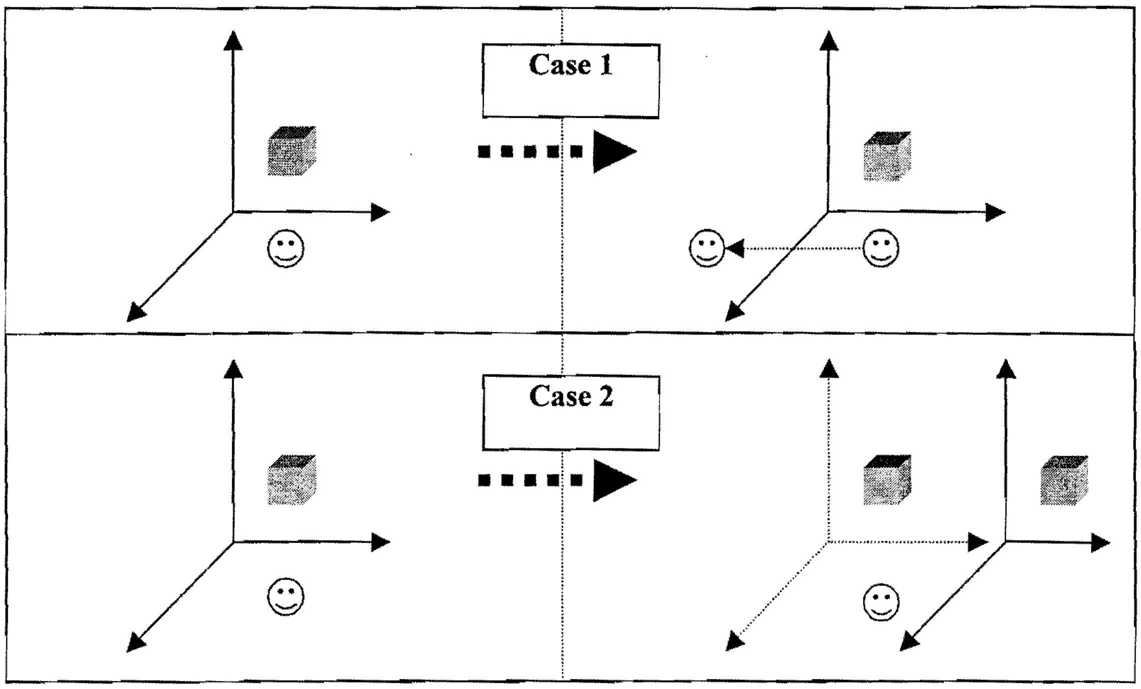


Figure 4.3: Creating the Impression of Movement

In case 1, the coordinate system and object in space remains fixed, and the viewer moves. In case 2, the viewer remains at a fixed position, and the coordinate system and the object move with respect to a static viewer.

If the scene moves, the coordinate system must rotate and translate with the object whereas if the viewpoint moves, the coordinate system remains fixed and the user rotates independently. Although there is no difference in appearance, it is sometimes considered not to be true 3D visualisation if the scene moves (case 2), since this does not normally represent reality. For visualisation of archaeological sites one would walk closer to the point of interest to have a closer look rather than moving the site towards yourself. This project has a default coordinate system which remains fixed (case 1), and the viewer moves relative to a fixed scene.

4.4.2 Static Scenes

As mentioned previously, a VRML file takes the form of a text file which describes a scene comprising objects of defined shapes and sizes to be placed at certain positions. Some of the elementary shapes can be called directly, for example cones and spheres. A scene can be created using these simple shapes, but more complex features must be defined by specifying polygons that best represent the feature.

VRML is in essence a file format used to describe a scene layout where the scene can comprise of:

- 3D data
- Audio
- Video
- 2D data (images)
- Text

It has predefined functions which are used to incorporate and position each of these data types into a single scene. These functions are called nodes, and they are the fundamental building block of a VRML file. VRML comprises 54 nodes, examples of some of the elementary nodes are:
"*Cone*", creates a regular cone.
"*Material*", which specify among other things, the colour and brightness of an object.
"*Transform*", determines the scale, orientation and position of an object.

A node contains fields which specify the parameters of the node. For example, the "*Cone*" node has a field called "*height*". As the name suggests, this sets the height of the cone (describing one of the five possible forms of data in a scene i.e. 3D data). A complete VRML file could be as simple as the following and yields the result shown above in figure 4.1:

```
#VRML V2.0 utf8
Shape {
geometry Cone {height 2}
}
```

Example 4.1: 3D Cone

All VRML files must start with the header "#VRML V2.0 utf8". This indicates to the browser what type of data to expect. The rest of the code is very simple to understand. *Shape* and *Cone* are nodes (they must begin with capital letters); *geometry* and *height* are fields. *height* is a property of *Cone*, similarly, *geometry* is a property of *Shape*. The code so far would add a regular cone with a height of 2 units at the default position to the scene.

The concepts introduced so far are that an arbitrary number of elementary shapes could be added sequentially to form the universe or scene. A more difficult concept to grasp is that of the relationship between different nodes. In this example, it is not possible to set the initial position of the sphere to a desired location without a link to the node responsible for positioning. Each node has a specialised task that it can perform but is limited if it is not combined with other nodes. Nodes must be nested within one another rather like logical statements are nested within one another rather like logical statements are nested within one another when programming. Ordering the tasks is important, as they form a hierarchical group where each of the internal nodes (labelled children) abide to the conditions set out by the children that are higher in the hierarchy than they are. Going back to the example above, if the cone is to be moved, the node responsible for positioning (*Transform*), must be used.

```
Group { children [
Transform {
translation 0 5 0
children Shape {
geometry Cone {height 2}
}
}]}
```

Example 4.2: 3D Cone translated

Here, the *Shape* node is nested inside the *Transform* node which will translate any of its children 5 units along the positive *Y*-axis. This is the basis for setting out a 3D scene, each node performs its task but abides to the conditions set out by any of the nodes that are higher in the group than they are. This principle is what makes VRML code portable. The code that represents the cone is identical in both examples; the code used for positioning the cone operates entirely independently. This enables one to copy portions of code from one VRML file and place them in the appropriate section of another file.

Becoming proficient in VRML is largely a matter of becoming familiar with the nodes that are available and how they combine successfully. The examples shown thus far can be extended to display a number of objects which make up a static scene. The user can now pan, rotate and zoom around the model viewing it from any desired angle. If the scene was that of the interior of a house, it would be possible to move from room to room but very little else. This is sufficient for some applications but for others, interaction with the scene is required. It might be useful that sensors could be set up that detect the movement of the user in the scene and automatically open a door or switch a light on when the user is close enough. Java is used to make VRML interactive and this will be covered later in more detail. VRML provides the framework which Java links to and must be understood before the two will combine successfully. The next task is to determine what makes it possible for Java to interact with VRML.

4.4.3 Interaction with the scene

Types of Behaviour

When designing VRML, proposals regarding the framework for making VRML an interactive 3D package were called for. A number of proposals were received, each fitting in to one of three categories; API, language and event-based (Lea *et al*, 1996).

API. The scene can be manipulated by a set of methods made available to the programmer. This approach will be familiar to most computer users. This can be thought of in terms of almost any modern program that uses a graphical user interface (GUI), where a set of procedure calls are made available which are accessed via a user interface. The advantage of this method is that there is no need to link to a separately programmed file that controls behaviour and in this way programming issues are simplified.

If this method were to be applied to VRML, it would result in the designer only having limited control over the actions that can be applied to the objects within the scene. Any behaviour that would be implemented would have to make use of the set of predefined functions made available. A visual framework is provided, and a set of functions are given to deal with the expected needs of the user. If these functions are insufficient, there is nothing the designer can do to account for this which limits the API approach.

A more pertinent problem is that it would not be possible to optimise performance because the scene does not contain any information regarding the intentions of the user. The lack of information arises from the fact that the data is clearly separated from the logic that manipulates the data. When using the API approach, the browser has no means of predicting the users possible actions. Functionality is also provided which may never be used which has a detrimental effect on performance. Other methods are tailored specifically for the expected needs of a user and as a result are more efficient.

Language. Using this approach, VRML would evolve to become a fully fledged, independent, specialised programming language (e.g. Java 3D) which would use shapes and objects as its basic control structure. This is entirely the opposite to the API approach because the 3D data and the behaviour of the data is combined (whereas they were totally separated in the API method). The language approach allows the Internet browser to have detailed information regarding the scene, including the behaviours that are likely to have to be implemented. As an example, the browser will now be capable of deciding which parts of the scene are outside the view of the user, and will therefore be able to render only the parts that are visible. The drawback to this approach is that the browser would have to have very complex specifications in order to deal with such demands. One of the constraints of VRML was to have a minimum demand on the browser, and therefore this approach was not adopted.

Event-Based. Events are used to send changes from an external language (Java) to the scene and vice-versa. (Lea *et al*, 1996) This is the approach adopted by VRML. In this method, the scene objects are entities which expose some of their internal states by means of fields. As an example, a *sphere* may expose its internal state such as it's radius or colour. This is somewhat of a compromise between the previous two methods. The 3D data is clearly separated from the language that controls its behaviour, but in this case a means of transferring information between the two is established.

The interface between VRML and Java is called the execution model. A trigger such as a mouse click activates the execution model which transfers data between the 3D scene description, and Java. Some logic is then performed by Java which returns the result to the execution model, which in turn is sent to VRML, resulting in the behaviour being performed. This is called Event-Based because the execution model transfers the data using "events". These events are simply messages containing data.

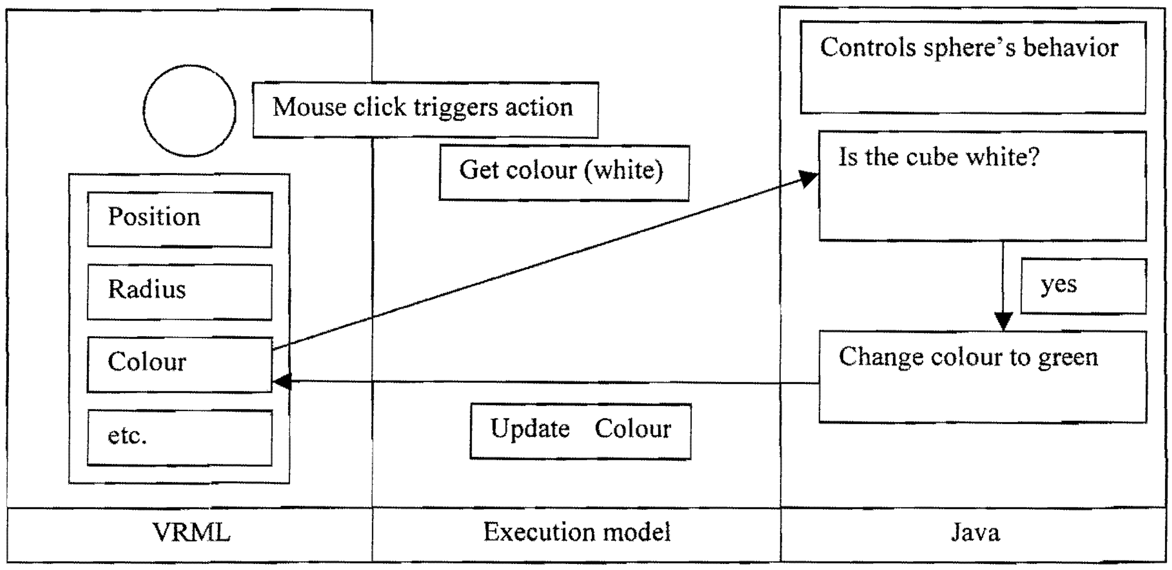


Figure 4.4: Execution Model

The event-based approach to a large extent separates the VRML file and the language (Java) which controls the behaviour of the scene. In the cases in which behaviour is not required, VRML can act entirely independently providing a very efficient means of viewing 3D scenes. If user interaction is required, the execution model acts as the bridge between VRML and a program such as Java. It also sets a strict set of rules to which transferred data must conform, thus ensuring it is passed in the correct format and eliminating any possible programming errors that could cause an error. The execution model defines which fields can be altered or manipulated and which are to be hidden. This allows for performance to be enhanced because there is a degree of control over the manipulation that can occur to the scene.

The Execution model

A definition is perhaps the best way to clarify the meaning of the execution model:

The execution model is an interface between VRML and the Java applet. This interface allows four types of access to a VRML scene (Landes, 1998)

- Getting notified when events from within the scene are sent.
- Reading the last value sent from within the scene.
- Sending events to objects within the scene.
- Accessing functionality of browser scripting interface.

When designing VRML, a strict set of rules was implemented specifying the extent to which the scene could be altered at run time. A trade off is made between allowing the user sufficient control over the behaviour of the model, and limiting other behaviour manipulation that would result in loss of performance.

Referring to Example 4.1, the height of the cone is by definition an "exposedField". This means that it is exposed to an external authoring interface (EAI), such as Java, or to the scripting function, i.e. JavaScript, and it is therefore possible to access those values and to change them at run time. Other possible field properties include "eventIn", and "eventOut"; as these names imply, eventIn means that the value can be changed, and eventOut means the value can be read. If neither of these properties is specified by the definition of the field, this field is hidden and cannot be seen or changed once it is defined in the original VRML scene layout.

In order to illustrate this more clearly, it is best to have an example. The detailed specification for the Viewpoint node is as follows:

```
Viewpoint {
EventIn          SFBool       set_bind
exposedField     SFFloat      fieldOfView   0.785398
exposedField     SFBool       jump          TRUE
exposedField     SFRotation   orientation   0 0 1 0
exposedField     SFVec3f      position      0 0 10
field               SFString      description    ""
eventOut            SFTime        bindTime
eventOut            SFBool        isBound
}
```

The first column indicates what access rights are allowed to each field. The second column is the variable type of that field, SFBool is a boolean value and so on. The third column is the name of the field and the last is the default value. From this, one can ascertain that full access of the position field (exposedField) is allowed, which enables that value to be modified at run time. For example, a button could be used to teleport the user to a new position in the scene by reading the current position value, incrementing its value, and then returning it. An exposed field is the most flexible field, followed by the eventIn field. Although this value is hidden for read purposes, it can still be modified. If the position was an eventIn field and we wanted to teleport the user, we could set the new position to a specific value, but would not be able to relate the position to the previous position since there is no read access allowed. If a field has eventOut rights, the value can be seen, but not altered, with a similar result of defining a file on ones computer read only. Finally, a field with no read or write privileges, is simply called field. Here we see that the description field has no access rights, once it's value has been set in the initial VRML code, it can never be changed or viewed by an EAI.

These limitations of read and write privileges are the result of the difference in design of VRML to other visualisation packages that have there own EAI. Java3D for example is similar to Java with additional classes and functions which enable viewing and manipulation of 3D scenes. There are no restrictions on what can and cannot be done within the Java file. This method typically produces more code than VRML but does allow for richer features. The fact that VRML puts restrictions on the use of certain nodes, means that it is more difficult to manipulate the data to suit ones own needs, but the written code is less and in general, rendering performance is better.

Summary

Thus far, the general principles of design of 3D models have been covered. VRML files are written in a text editor, and are interpreted by the browser. A scene is created by adding nodes and setting the fields of those nodes. An effective scene is created by nodes being grouped together in such a way that their functionality is shared. Finally the concepts for scene interaction have been introduced. This topic will be continued when dealing with JAVA.

It was previously mentioned that VRML was not strictly a modelling language since it does not contain some of the basic modelling primitives. VRML does however have nodes which are very powerful and far extend normal modelling capabilities. Some of these will be briefly discussed.

4.5 Advanced VRML

The title of this section may intimidate some readers, but there is no need for this. Advanced VRML does not necessarily mean that it is difficult, but rather that the methods in this section are used to improve the functionality and efficiency of a basic VRML scene, and thus enhance the basic VRML file. These topics are more interesting and less theoretical than the basic theory which has been discussed up to now. This section is divided into five sub-sections, each dealing with a specific node that is used to perform an advanced function. These nodes are used to extend the basic nodes such as shapes and colours.

4.5.1 Level of Detail (LOD) Nodes

In reality, as one moves closer to an object, one will perceive more of the objects visual detail. The LOD node accounts for this by allowing for different levels of detail of the same object to be shown, depending on the distance the user is from that object. This node improves the programs efficiency significantly because unnecessarily complex shapes do not have to be rendered if they are too far away to be perceived. A complex shape made up of thousands of vertices can be simplified and represented by a single point (or even ignored totally) if it is far enough away.

If an object is within the field of view, even if that object is so far away that it cannot be seen, all rendering calculations still have to be performed. This can often lead to a vast number of unnecessary calculations being performed for a very distant object that represents a very small part of the screen. VRML's part in the LOD operation is to determine when the threshold distance is reached and then to replace the existing representation of the object with the new one. The threshold distances are normally specified manually and the alternate representation of the objects are always specified manually. In the case of the footprints, the files of lower detail were made from the same DEM using every second, third or fourth point (each time detail is lost) in each direction creating separate models for each level.

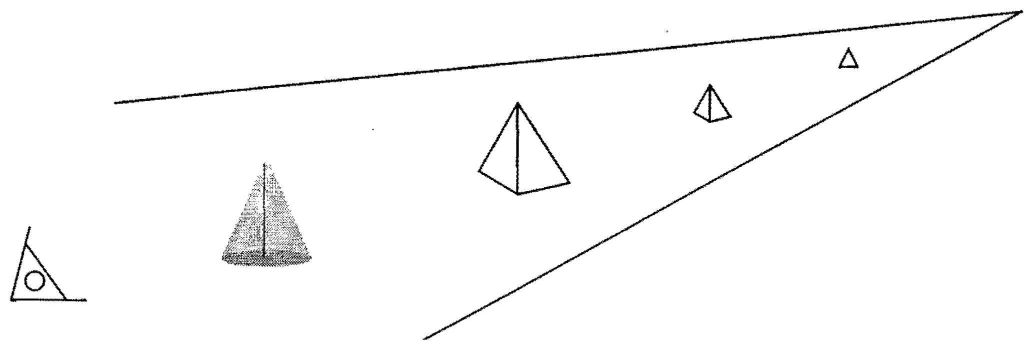


Figure 4.5: Level of Detail

Specifying the level of detail and the distances at which switching takes place must be done very carefully so that the viewer cannot detect that the object has been replaced by a more simple representation. The ideal switching distance between levels is that exact distance at which the viewer cannot detect the change when moving from one level of detail to the next. This can be calculated by assuming the resolution of the users monitor and working out how many pixels an object will comprise of at a certain distance. From this, an estimate of the required complexity of the object can be made.

EXAMPLE 4.3: Level of Detail

Some assumptions need to be made, and in order to keep matters simple, the following is assumed:
Screen Resolution: 800 * 600 pixels;
Higher Level of Detail DEM: 80 columns, 60 rows, at a 1 unit spacing;
Lower Level of Detail DEM: 40 columns, 30 rows, at a 2 unit spacing;
Field of View: 45°.
Human vision: A further prediction has to be made of the ability for the human eye to perceive small variations in colour. For the purpose of this example, it is assumed that the human eye would not perceive a difference in colour of a pixel spanning 2 DEM point, to another pixel spanning 3 DEM points. This is likely to be the case because adjacent DEM points will not be represented by vastly different colours under normal circumstances.

The question to be answered can be phrased as follows:
At what distance will the object of lower detail be represented by exactly 1 pixel for every 2 DEM points? (Making this representation adequate to replace the higher level of detail). If the object of lower detail is any closer than this threshold distance, a pixel will span less than 2 DEM point, and the difference in appearance will be noticeable.

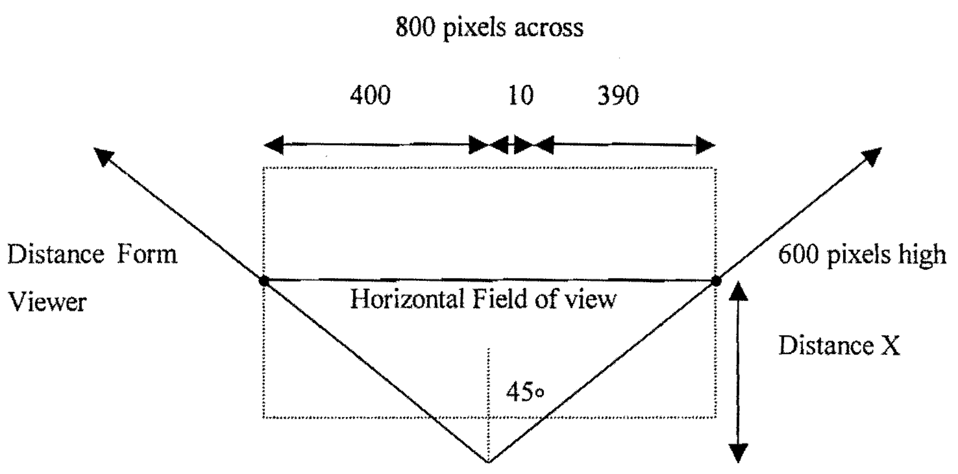


Figure 4.6: View of Monitor for Calculating LOD

10 Pixels should represent 20 DEM points (which is equivalent to 400 pixels representing 800 DEM points). The DEM spacing is 2 units, X is the distance required, and from simple geometry:
$\tan(45°) = 1600/X$,
$X = 1600$ units, and this is the distance at which the user would not distinguish between the two levels of detail of the same object.

This function is one of the most effective methods for increasing the speed of rendering. The slight disadvantage to using this method is the slightly larger size of the files that need to be downloaded, since the objects of less detail have to be specified and included along with the original object.

4.5.2 Inline Nodes

When designing a scene, it is common that existing models created locally or found on the Internet need to be included in the new scene. Including this existing data can be done by using the *Inline* node which works by means of referencing that model in the new scene in the form of a URL. This feature is a natural extension of VRML because of its tight integration into the Internet.

This method of calling independent VRML files using the *Inline* node is one of the biggest strengths of VRML. It is possible to create a complex VRML file without defining a single shape or object, but rather a list of locations of existing objects referenced by means of URL's. It is likely to be difficult to regularly find a 3D object on the Internet to suit ones particular needs, but the size of existing 3D libraries is increasing rapidly, particularly in the field of engineering applications. It is more likely that one would want to include files created locally in new scenes. A number of objects defined separately can then be used in any number of combinations with a minimum of effort.

The borrowing of existing 3D scenes in this way has potential for very powerful applications to be created, especially considering the improvements in hardware performance. This will be discussed further in chapter 4.6 (The future of VRML).

4.5.3 Prototype Nodes

This node is often confused with the *Inline* node, but is used in slightly different circumstances. Often, a combination of nodes is set up and needs to be used a number of times, each time with only slight changes. This is when prototyping is used. For example, if one wanted to create a 3D model of the computer equipment in an office, one would need to have a 3D representation of a computer set up at each desk. If there were 4 computers, it may be convenient to represent each with a different colour. It would be very inefficient to use the same code 4 times in a single file, each time changing a single colour field. When using prototyping, a separate file containing the shape and dimensional description of a single computer could be made (a prototype). For each of the four computers in the example, reference can be made to the prototype, with the option of customising that prototype to suit ones own needs.

This method is very powerful and along with the *Inline* node, is the key to keeping VRML files relatively small and manageable. In landscape scenery, it is often the case that many very similar objects need to be used. Trees are the most common example where a forest must be defined by a large number of very similar looking trees. Once again, a standard tree used to represent the norm is defined as the prototype, and the appearance of a real scene is created by placing a number of prototypes close to one another, and by changing the colour and size of a random scatter of those objects.

Having said that VRML files are kept relatively small, it is not to say that VRML files are small in general. It means that the files would be a lot bigger without the use of these methods. The size of a file can be minimised if there is data that can be repeated (and thus defined by using prototypes and inline nodes). VRML has efficient ways of managing the data, but the fact that they are ASCII files makes still means that they are larger than compiled files. A large amount of space is wasted with unwanted data such as space characters.

4.5.4 Sensor Nodes

As the name suggests, sensors detect activity and respond accordingly. In the examples of the computer arrangements in the office, one could set up a Proximity sensor that announces the owners name when the user navigates within a metre of it by triggering a *.wav* file.

Examples are endless, and are left up to the imagination. Rather than discussing possible uses, a list of some types of sensors, each with a simple example of their possible use are given below.

TimeSensor. Animations are often performed by means of specifying a number of positions of an object, which are set into motion when a timeSensor triggers them off. Such a set of positions could be a set of ten frames defining the motion of a cyclist pedalling his bicycle. The speed at which the animation takes place is determined by the time sensor (This is known as keyframe animation).

TouchSensor. Touch sensors detect a mouse click with on part of the geometry of a scene. This could be used to click a light switch to turn it on or off.

VisiblitySensor. This is used to speed up processing, when the results of a process are not visible to the user. If the viewer was not able to see the keyframe animation (e.g. if it is obstructed) mentioned above, the unnecessary details do not have to be calculated.

4.5.5 Script Nodes

The scripting node allows foreign code to be included in the VRML file, allowing for another language to deal with the scripting functions. Although it is possible to use any programming language to control behaviours, JAVA and JAVASCRIPT are the only two practical options due to their tight integration with the Internet. Typically, user interaction is made possible by setting the framework using nodes, sensors, and interpolators, written in the VRML file. This sets up the initial scene but once it is in use, scripts must control the more complex user interaction.

Scripts act as the bridge between VRML and Java extending the capabilities of VRML making it possible to create sophisticated interactive worlds.

4.5.6 Summary

This section has outlined some of the more interesting nodes which make up VRML. These and other nodes are used to add to the basic static structure of VRML 1.0, which can now make up interactive and efficient 3D scenes. The most important addition from VRML 1.0 as that of *Scripts* and *Sensors*, which are largely responsible for user interactivity. Using these tools, VRML is now capable of performing advanced functions providing the user with a more realistic experience. The future of VRML is largely dependant on the designers of VRML scenes and the results achieved.

What is in store in the next generation of VRML? The following chapter takes a look at some likely additions to VRML, as well as some developments which are expected to effect VRML.

4.6 The Future of VRML

It is difficult to speculate the future of any software especially when considering the rapid hardware developments. It is nevertheless interesting to consider some of the short term development possibilities for 3D visualisation and VRML in particular.

Since 1994, VRML has become the standard of 3D visualisation on the Internet. It is apparent that the future success of VRML is reliant more on the growing number of users rather than the original developers (Carey & Bell, 1997). VRML has some areas in which it is expected that the first improvements will take place.

4.6.1 Possible additions to VRML

Binary File format

As mentioned above VRML files tend to be larger than they need to because they are in ASCII format. At present, this problem can be overcome to some extent by using compression utilities such as gzip to compress the files before distributing them. Work is being done on defining a binary format which will enable efficient conversions from ASCII format for more efficient transmission on the Internet.

Not only would this reduce download time, but it would also enable security to be more efficient. At present, the code for any VRML file on the Internet can easily be accessed, which results in some reluctance to distribute files on the Internet (Lea *et al*, 1996).

Multi-user Capabilities

This is likely to be one of the main additions to a future version of VRML. This technology would enable two or more remote users to view and interact with the same scene at the same time. Any changes that are made by any user affects all other users. It has been suggested that the notion of presence will have to be introduced, where every user must be represented by what is called an avatar, i.e. a 3D representation of the user which moves where the user does and carries out their commands.

4.6.2 Improved and New Technologies

In order to create a visually appealing moving scene, a frame rate of 10 frames per second is sufficient (Hartman & Wernecke, 1996). At the time this handbook was written, the general rule of thumb was that this implied a maximum polygon count of roughly 2000. This figure depends entirely on the hardware used, and thus this number will increase as hardware improves. A frame rate of at least 18 is required for video and if this frame rate could be achieved in visualisation applications it would result in very smooth movement and a greater appreciation of the product. It is likely that this figure will be the norm in the near future.

Although it is impossible to estimate the future developments in hardware performance, it is interesting to note some of the developments in the last few years. It gives an indication of the progress made and helps to provide some idea of the forthcoming advances in technology. The components that most affect 3D visualisation are:

Graphics cards

Graphics cards are highly efficient devices designed specifically for creating the images that are to be displayed on the monitor. This process was once dealt with by the main processor, but when graphics demands increased, it became necessary to pass these calculations to a type of co-processor which lead to the development of the graphics card.

The following table shows the main advances in graphics card used for 2D graphics (Anderson, 1999).

Table 4.1: Video Card Technology

Year	Standard	Description	Resolution	No. Of Colours
1981	CGA	Colour Graphics Adapter	160 x 200	16
1984	EGA	Enhanced Graphics Adapter	640 x 350	16 of 64
1987	VGA	Video Graphics Array	640 x 480 320 x 200	16 of 262144 256
1990	XGA	Extended Graphics Array	800 x 600 1024 x 768	16.7 million 65536

In recent times, the 3D graphics card has been developed which, as the name suggests, calculates information in a 3 dimensional coordinate system, dramatically improving the performance of 3D visualisation (Anderson, 1999). It would be a major task to attempt a detailed investigation into 3D Graphics cards since there are so many different types available, and the performance of video cards is measured in various different ways. As an example however, the Creative Labs - RIVA TNT Graphics Blaster (1998), is able to calculate up to 6 million triangles per second (Creative Labs, 2000). This gives an indication of the assistance that the graphics card assists 3D visualisation. These calculations which would once have been performed by the processor are now shared with specialised hardware.

Processor speed

The processor affects almost all aspects of a computers performance. In general terms, the quicker a processor is, the more calculations it can perform per second. Processor speeds have increased by roughly 50% every year since 1994 (Burd, 1999).

Table 4.2: Release Dates of Various Intel Pentium Chips

Date Released	Processor Speed (MHz)	Pentium Chip
1994 (March 7)	100	I
1995 (March 28)	120	I
1996 (June 11)	200	I
1997 (May 7)	266	II
1998 (August 24)	450	II
1999 (February 26)	550	III

A test on a local computer shows that when rotating an object of roughly 12000 points, the processor is working at 100%. This is perhaps expected, but what was surprising is that the same test done on a DEM comprising of 20 points yields the same result. The processor works at its maximum rate regardless of the complexity of the model in order to achieve a maximum frame rate. This suggests that any improvement in processor speed will directly affect the performance of 3D visualisation. This will not necessarily be the case for all configurations because at some point a hardware component will be operating at it's maximum speed and therefore create a limit on the speed that information needs to be processed by other devices (sometimes referred to as a bottleneck).

Although it is difficult to attach precise numerical figures for an improvement associated with each hardware device, video cards and processors make the largest impact. Advances in both these technologies show no sign of slowing down. Intel processors are now running at 850Mhz, and 32Mb graphics cards (used exclusively on specialised workstations a year ago) are now standard. The future of technology improvements will result in very detailed models being practical for use in 3D visualisation and an improvement in frame rates making for highly visually appealing models.

Internet connections

VRML is designed for use over the Internet and it is to be expected that improvements in any Internet technology will have a positive effect. Although improvement in Internet access speed does not affect the performance of rendering, it does allow data to be gathered more quickly and efficiently. It is often the case that the time spent waiting for a file to download is more critical than the frame rate when viewing the file. It is more difficult to establish current standards of Internet connections since there are many different methods for downloading, and the fact that the Internet is relatively new. It is apparent however, that competition for this market is strong, and improvements are rapid. New trends in satellite connections are emerging and the results of this form of transfer are likely to outperform traditional methods.

4.6.3 Other Technologies

Continuous Media

Of the data that can be included in a VRML file, continuous media are the least developed. Continuous media includes streaming audio and streaming video. The ability to deliver audio and video across the Internet is becoming more efficient. Developments of continuous media are likely to result in better means of integrating this data into VRML files.

(Lea *et al*, 1996)

Web Television

South Africa has already seen the introduction of Internet on satellite television. It has been predicted that most Internet surfing will be carried out using TV, simply because more users feel comfortable using operating the TV rather than a computer (Lea *et al*, 1996). VRML is expected to be used a great deal by TV web users because interactive 3D graphics will be well suited to large screens.

Chapter 5 Processing Primary data

5 .1 Introduction

The steps involved in the creation of an accurate computer visualisation of an object are, collecting data pertaining to the object, checking the data, and converting the data into the required format. By the time this project began, the data collection had been carried out in two field campaigns and the following data was available:

DEM's – Each footprint was described by a DEM at a grid interval of 2.5mm. The files comprised of an *X,Y,Z* coordinate list of approximately 12 000 points,

- Ortho-images – An ortho-image for each DEM,
- Contour Files – A 1mm contour file for each footprint (Autocad/DXF format).

Editing programs are needed to convert raw data into the required format for visualisation purposes. The first part of the section briefly lists the editing programs written for converting raw data. The remainder of the chapter takes a more in-depth look at how the data was checked which is a very important stage in visualisation and is often neglected. There is a need for a systematic approach to error control for data used in Visualisation (Brodlie *et al*, 1992). Gross Errors introduced in the process of recording and processing data need to be eliminated so as not to introduce a bias when the final result is interpreted.

5.2 Conversion software

Once the framework for the VRML file was established on a small test case, a program was required which would arrange larger data sets into this format. The programming language used to create most editors was Visual Basic – a high level programming language that is well suited for such tasks where performance is not a major concern. Although Java is also slow, it has the ability to handle much larger data sets and for this reason it was used in the cases where data sets consisted of more than 30000 coordinates. DXF files in particular are likely to be larger than this threshold and Java was solely used for their manipulation.

5.2.1 DEM's

DEM data was available in the form of a list of XYZ (standard deviation of 0.4mm in X,Y and 0.6mm in Z) coordinates representing points on the surface of the footprints. A single Visual Basic program reading this file was created to produce the following VRML files:

- Point Scatters,
- Wiremesh grids (square or triangular),
- Solid surfaces with colour graduation according to height,

Chapter 5 Processing Primary data

5 .1 Introduction

The steps involved in the creation of an accurate computer visualisation of an object are, collecting data pertaining to the object, checking the data, and converting the data into the required format. By the time this project began, the data collection had been carried out in two field campaigns and the following data was available:

DEM's – Each footprint was described by a DEM at a grid interval of 2.5mm. The files comprised of an *X,Y,Z* coordinate list of approximately 12 000 points,

- Ortho-images – An ortho-image for each DEM,
- Contour Files – A 1mm contour file for each footprint (Autocad/DXF format).

Editing programs are needed to convert raw data into the required format for visualisation purposes. The first part of the section briefly lists the editing programs written for converting raw data. The remainder of the chapter takes a more in-depth look at how the data was checked which is a very important stage in visualisation and is often neglected. There is a need for a systematic approach to error control for data used in Visualisation (Brodlie *et al*, 1992). Gross Errors introduced in the process of recording and processing data need to be eliminated so as not to introduce a bias when the final result is interpreted.

5.2 Conversion software

Once the framework for the VRML file was established on a small test case, a program was required which would arrange larger data sets into this format. The programming language used to create most editors was Visual Basic – a high level programming language that is well suited for such tasks where performance is not a major concern. Although Java is also slow, it has the ability to handle much larger data sets and for this reason it was used in the cases where data sets consisted of more than 30000 coordinates. DXF files in particular are likely to be larger than this threshold and Java was solely used for their manipulation.

5.2.1 DEM's

DEM data was available in the form of a list of XYZ (standard deviation of 0.4mm in X,Y and 0.6mm in Z) coordinates representing points on the surface of the footprints. A single Visual Basic program reading this file was created to produce the following VRML files:

- Point Scatters,
- Wiremesh grids (square or triangular),
- Solid surfaces with colour graduation according to height,

- Solid surfaces set up for image (preferably ortho-image) overlay.

The same program also allows some properties of the resulting VRML file to be set:

- Choices of background colours,
- Setting of viewpoints,
- Reducing the number of points used,
- Setting height exaggeration.

5.2.2 Contours

Dealing with contour files was more difficult than the DEM files because the data was only available in DXF format. An additional step was needed to convert the large DXF files into a smaller DEM format. A Java program was written to reduce the number of points in the file by approximately 90% making it practical for visualisation purposes. There were no mathematical criteria for deciding on the number of points to use for the contour lines, it was a matter of including enough points that the contours appear smooth. If too many points are excluded, the appearance of the contour lines may appear jagged and this must be avoided. The final step in this editing process was to convert the resulting DEM into a VRML line drawing (using Visual Basic).

5.3 Error Detection

5.3.1 By Inspection

This section is concerned with the detection of errors in the data used. It does not attempt to deal with issues regarding general accuracy but rather with detecting and eliminating outliers or blunders. There are three types of data used for visualisation in this project that need to be checked, namely DEM's, contour files and ortho-images.

At different stages of the project, each of these forms of data were dealt with independently and viewed as a VRML file (Chapter 6). The first data type put to use were the DEM's which were used to produce the following:

- 3D Point scatters,
- Triangular or rectangular wiremesh grids,
- Continuous surfaces representing the footprint and surrounding region.

It is assumed that the DEM files are checked when viewed in 3D format because any obvious blunders would be noticed by inspecting the product obtained in VRML. Results obtained from each of these steps above were VRML files that could be viewed from any desired angle or distance making inspection of the data more thorough. It is difficult to notice any inconsistencies when viewing the point scatter, but the grids and continuous surface in particular form an image in the minds eye which is sufficient for a basic check for blunders. Consider figure 5.1 which shows a spike in a continuous surface derived from a DEM. Although this error would most likely not be noticed if the data was shown in its basic form of a point scatter, the creation of a continuous surface means that visual data connecting the remaining points to the outlier highlight the error.

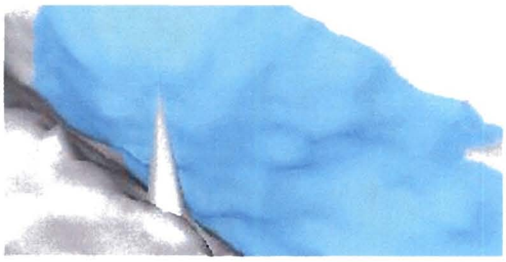


Figure 5.1: An Erroneous DEM Value

Ortho-images are checked when they created to ensure they are geometrically correct. Accurately measured control points are used in the data acquisition process, and can be compared to screen coordinates on the resulting ortho-image that is produced. There are various possible errors that can occur in the creation of ortho-images such as image distortion and incorrect camera orientation parameters but this is beyond the scope of this thesis. Ortho-images are created by using the DEM points and if their are no blatant errors on the corresponding DEM the ortho-image is likely to be free of blunders. It was assumed that if the DEM's did not contain blatant errors, that the ortho-images were also error free.

Contours are more difficult to check than the other two forms of data. Some typical contour errors include missing contour lines, contours not closing and contours in the wrong position. Obvious errors such as contours crossing each other would be very difficult to notice in 3D form because contours do appear to cross each other when viewed from certain angles. Less obvious errors such as contour lines not snapping together correctly are even more difficult to notice. Although the contour files from the footprints cover such a small area, it is still not practical to check for errors by inspection because the contour is 1mm which results in a large amount of contour lines. From this, stemmed the need for a program to check for errors in a contour file.

A contour line is described in a DXF file as a set of coordinates of points along the path of the line. A single contour may be made up of hundreds and sometimes thousands of points (vertices). The number of points describing an entire contour file can run into the millions which makes traditional programming techniques (where data is stored in arrays) impractical. An alternative approach is used whereby data is separated into smaller categories and stored in files rather than arrays. The nature of the contour description effects the way the software is designed. Further details of the programming technique is discussed later.

The contours for the Laetoli footprints were generated automatically. The processes involved were:

- Random point generation,
- Point matching,
- DEM generation,
- Contour line interpolation.

5.4 A Program for Checking Contours files

5.4.1 Design objectives for the program

- Read DXF contour files. (considered an acceptable standard),
- Read files of up to 200Mb so that large regions can be checked,
- Accept "roughly" rectangular contour files,
- Work with any user defined contour interval,
- Check for interpolation and digitising errors,
- Run on a computer as a background process, without interfering with other operations,
- Run overnight without crashing. It has to be very stable,
- Write new DXF files,
- Write report files on changes made to the new DXF file, for post checking.

5.4.2 Types of Problems to be expected

Assuming a contour interval of 20m (or 1mm in the case of the footprints), some expectations can be made about adjacent contours i.e. that if they are not exactly equal or differ by 20 metres (in height) , then there is an error involved. This basic fact forms the basis for checking the heights of contours. If a contour has the correct height value, it does not mean it is free of error since there are other possible errors involved (e.g. contours not snapping together correctly or the position of the contour being incorrect) . Below is a list of some of the possible errors that can occur:

1. Jumps in contour values that can be logically corrected e.g. 600, 620,*740*,660 – Figure 5.2, A;
2. Errors that have no logical correction e.g. 520, 540,600,600... arising from contour lines being skipped. This was found to be a regular occurrence on steep terrain where contours are very closely spaced – Figure 5.2, B;

3. Contour lines not joining correctly at their terminals (snap error) – Figure 5.2, C. The area shown as – Figure 5.2, C* is a special case which can be mistaken for a snap error when there is no error;
4. Contours being made up of more than one polyline. This occurs in cases where digitising is interrupted and then continued at a later stage. (This problem does not occur from computer generated contours. This error occurs during manual digitising and was considered so that the program could be used to check manually generated contours). This does not necessarily appear to be an error when viewing such a file (if the join between the old and new line is made correctly) but it leads to problems for a number of reasons. The first is that small line segments are introduced in this way which often have erroneous height values i.e. the contour value changes along a contour line. The line segments also introduce problems because open polylines should only occur on the border of a contour region whereas these line segments are also open polylines and they terminate anywhere within the contour region. - Figure 5.2. D;
5. Other errors such as contours crossing each other – Figure 5.2, E.
6. Errors that cannot be corrected such as inaccurate interpolation which does not follow the true path of the contour correctly.

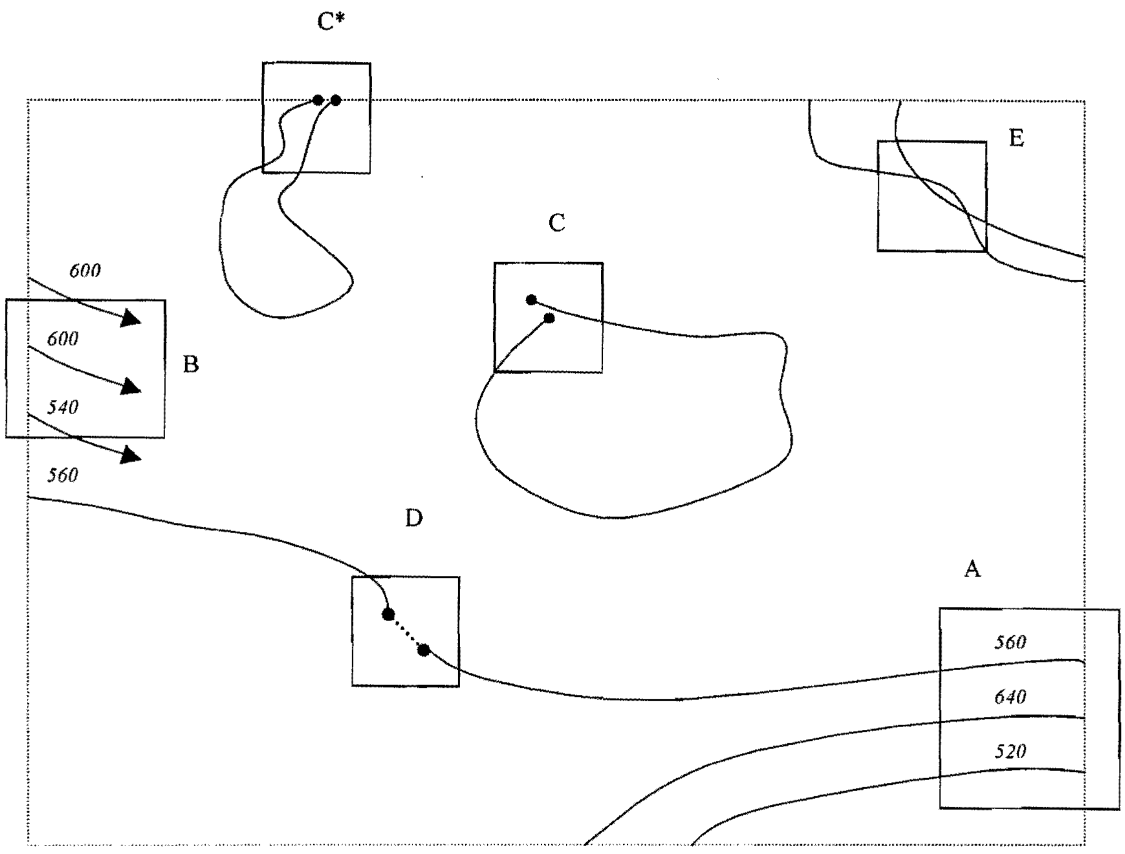


Figure 5.2: Contour Lines at a 20m Interval

5.4.3 General Approach

It was decided that the problems listed above should be separated into three categories. Errors in cases A should be corrected automatically as there is a logical correction that can be performed. Errors in cases C and D should be pointed out for manual correction as a pre-processing stage, and errors B and E should be pointed out to the user at the end of the process for post processing checks. The reason for this separation will be made clear in later discussion but is essentially determined by dividing tasks that require human interpretation and those where a clear-cut mathematical correction can be made.

Before looking at any of the technical details, an understanding of the problem is required. The two underlying difficulties are:

Sorting the contours into a logical sequence

If one was to check a set of contours by inspection, the natural way to check for errors is to compare the height of a contour to its closest neighbour on either side, and if this was inconclusive, by comparing the heights of other nearby contours. Human perception easily enables one to locate nearby contours and ignore those further away. A major difficulty in designing a program that checks the contours is to establish which contours lie on either side of a given contour.

The DXF file describes each line as a consecutive set of points in order along the contour. Consecutive contour lines stored in the file are not ordered in any way though so it cannot be assumed that the first and second contours lie next to each other. A number of procedures were ultimately combined in establishing this required relationship which forms the bulk of the problem.

Knowing which contours are correct

Once the contours have been ordered into a natural sequence, how does one determine which heights are correct? There needs to be some means of referencing a discrepancy to a "true" height, if corrections are to be made. If a "true" set of contours was not known, inconsistencies in the data could still be located, but corrections to this data could not be made. Consider the following example where a sequence of contour heights at a 20m interval is 140, 160, 280, 300… It can be seen that there is an error involved but there is no way of determining which two heights should be corrected. Now assuming it was known that the first coordinate in the sequence was correct, it would allow the remainder of the sequence to be corrected and would also mean that these checked and corrected contours could be used as a reference in further sequences. (The contours are now correct because at this stage in the programs execution, the other errors, such as snap errors have been dealt with).

Contour lines that terminate on the borders of the contour region are naturally suited to being separated in the assignment process from the remaining contours and regarded as more likely to be correct. The reason for this will be discussed further (contour files are often edge matched). Establishing a set of "true" heights was the first logical step in checking the file for errors.

5.4.4 Detailed Approach

Step 1: Reading the DXF file

Reading DXF files is a somewhat difficult task since there are so many different types of codes and characters used. DXF is a CAD drawing package used for a large range of drawings rather than for contour files alone and many of these codes are not used in the contour files. Specialising the program to read in only contour files eliminates the need to process all the possible codes supported by the DXF format simplifying the task somewhat.

DXF files are not always written in the format as set out by the documentation of DXF, for example it was noticed that ArcView sometimes writes the height value in the field assigned to contour line descriptions. Various difficulties such as these result in the need for user input in order to decipher each. The program reads the first point of the file and prompts the user to identify the *X*, *Y*, *Z* values (at this stage, the contour interval is also entered). This ensures that the correct fields are accessed for the extraction of coordinate information.

Step 2: Point out errors for pre-processing

Further description of the process requires some terminology to be well understood. Contours are described by a set of points and all contours are also polylines. Two type of polylines exist namely:

- *Polygons.* Which form a completed shapes, i.e. polylines which start and end at the same point,
- *Open polylines.* All other contour lines. This includes lines which terminate at the border of the region, lines which do not snap together correctly and lines made up of more than one continuous polyline. Refer to figure 5.2 (A – E) for example of each of these cases.

Snapping Contour Lines

Polylines that start and end very close to each other pose a problem. Consider a program that locates the terminals of a polyline and highlights the ones where they are close to each other – less than a certain distance specified by the user at run time. The program would locate areas such as C, C* and D from figure 5.2. C is the standard snapping error which is very common and can be dealt with. Areas C* and D are less common and are not necessarily errors. The program is able to disregard area C* because an edge tolerance can be specified to rule out possible snap errors ending close to the borders of the region. Area D is a more problematic as the dotted line represents a complete polyline and thus mistaken for an incomplete polygon, i.e. mistaken for case C. What the program does is locate these areas for a pre-processing stage before the rest of the check continues. The user is able to continue if they wish without manually correcting these areas, but then all these areas will be assumed to form polygons even if they shouldn't as in case C* (an imaginary line is added – the line is not actually drawn but the program considers it to be there so that calculations are performed as if it were a complete polygon). This makes accepting the snapping default a risky process. It is advisable that the areas of type D be corrected manually (made into a single polyline) to avoid possible errors. In the two cases where the program did not complete successfully (mentioned at the end of the chapter) it is expected that the problem was that this pre-processing was not carried out on an area of type D. Automatic correction is not always possible because the break in the contour sometimes has a different value to the remainder of the line.

There are various difficulties associated with snapping contour lines which make it a non-trivial task. Specialised programs (e.g. TNT MIPS) have routines which attempt to perform this task with varying degrees of success. Most programs take a very long time to run and often join contours which should not be joined, sometimes joining contours that run very close to one another. It was decided it would be better to produce a coordinate file (XY only – locating the terminals) which could be overlaid on the original contour file for manual checking if necessary.

Errors on the Border

Edge matching errors are the second category corrected in the pre-processing stage (Figure 5.2, D). It may seem confusing that this error is not corrected automatically when there is a logical solution. If this sequence of contour heights was located within the interior of the contour region, it would be corrected automatically but an extra check is required for the contour lines terminating at the border of the contour region because these form the basis for checking the remainder of the file (discussed in section 5.43 – knowing which contours are correct). This error should be not be encountered frequently because edge matching to adjacent regions is normally performed. In some cases errors are found though and are located using the following procedure:

Determine the shape of the contour region for the purpose of creating the buffer in figure 5.3 . The *X,Y* coordinate extremes can be extracted from the DXF file, but there is no immediate way of determining if the "rectangle" is tilted at some angle. The term "rectangle" is not entirely correct because the corner points are often not joined by straight lines due to a curve formed by the projection. All terminals are used in a calculation to determine the approximate shape of the border which is made up of four regions (one for each side) described by a straight line with a buffer to accommodate for curvature of the projection.

All terminals of the open polylines should now lie within a small border area determined above (all polygons are ignored). This is used to create a logical order of open polylines because it is possible to order ones that terminate at the "top" or "right" of the region and so on.

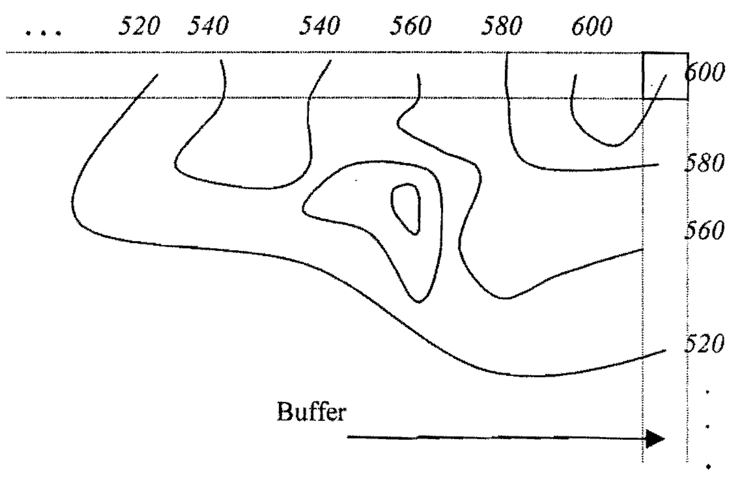


Figure 5.3: Checking Edge Matching

The buffer around the known border is split into four categories for each side which are used to record any terminals of polylines ending within them. The shaded block is considered part of the top buffer and not the right buffer to avoid a terminal being recorded twice. All the positions of the terminals are recorded and ordered into a logical sequence. For example, all the terminals ending in the top buffer are ordered left to right, and all the terminals in the right buffer ordered top to bottom. From figure 5.3, a sequence for the top (...520, 540, 540, 560, 580, 600, 600) and right sides (580, 560, 520...) would be obtained and then combined to form a long chain of height values. The same applies to the bottom and left sides respectively until all points in this buffer are linked back to the top left terminal. Now that an order is established the height values of adjacent terminals can be compared. A difference of adjacent values must equal zero, or the absolute value of the contour interval otherwise it is incorrect. Any errors are added to the text file containing the snap errors from above for checking in ArcView.

The result of overlaying the errors (snap errors and edge matching errors) onto the contour file are shown below – Figure 5.3 (left). In this case all errors are internal indicating that the edge matching was successful. This is most often the case, almost all errors occurring internally. A closer view of some of the errors reveals the types of inconsistencies located - Figure 5.4 (right). In this case, it is not essential to make any changes before the process continues. Edge matching errors must be carried out but snap errors do not have to be corrected although it would be preferable to do so. All three cases would be treated as if they were snapped together, but in the bottom case (a line that should be deleted), assuming the line was out of the range of the snap tolerance it would be treated as an open polyline. In this case, the checking process would not continue until that error was corrected because open polylines are used for checking the remainder of the file, and an error like this could affect it.

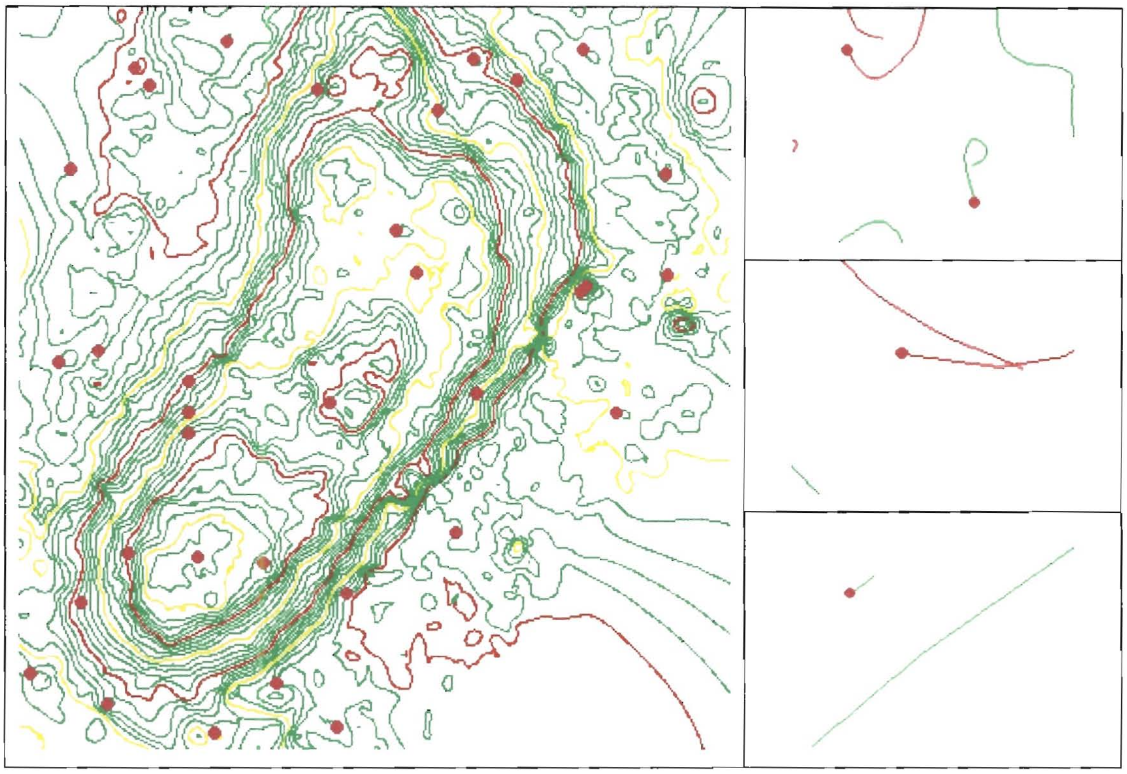


Figure 5.4: Errors Located for pre-processing

Step 3: Sort the data in to a useful format for referencing it. (Data files)

When dealing with such large amounts of data, it is not possible to store all the information necessary in memory. There can be literally millions of points comprising a contour file and all this information cannot be stored in arrays when running the program. A method of managing the data had to be created and it was decided that the necessary information would be stored on the hard drive in various forms so as to eliminate the need for huge arrays. The more logical divisions of data that could be created, the more efficient the process would be because the data would need to be referenced many times, thus searching for data (e.g. height values) from a file that includes data that is of no interest at the time (XYZ values) would be inefficient. Files are created for storing the necessary information when the program is run and contain details of the following:

- Open / closed polylines (separately),

XY / *Z* coordinates (separately),

- A file comprising the extreme XY coordinates of each closed polygon,
- A file containing the areas of each polygon calculated when reading DXF file initially.

Step 4: Create a sequence for closed polygons

Once again, as in the case of the open polylines, a sequence is needed if height comparison can take place. All open polylines are considered correct and will not be changed from this point on. A reference for the ordered polygons is obtained by locating the nearest open polyline. Figure 5.5 shows the desired result. The contour making up the largest area is located (1), and all polygons lying within this region are also ordered (2, 3, 3). The closest true value is found by locating the closest open polyline to the extreme left point of the largest polygon.

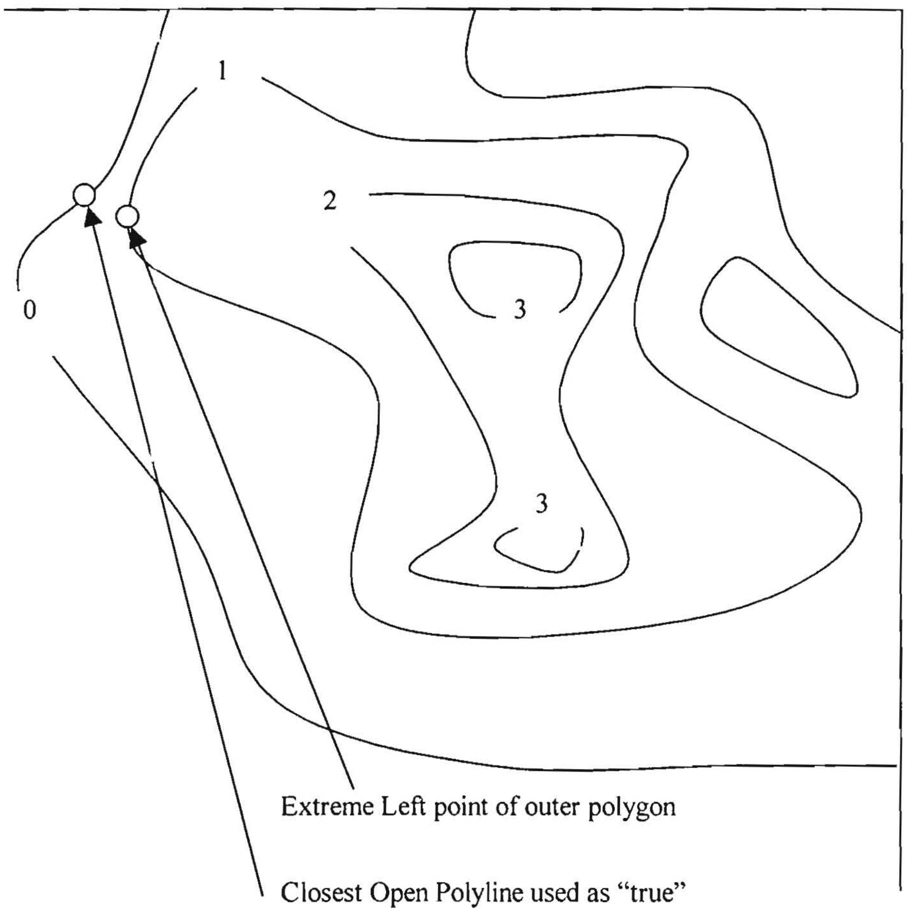

1
2
0
3
3
Extreme Left point of outer polygon
Closest Open Polyline used as "true"

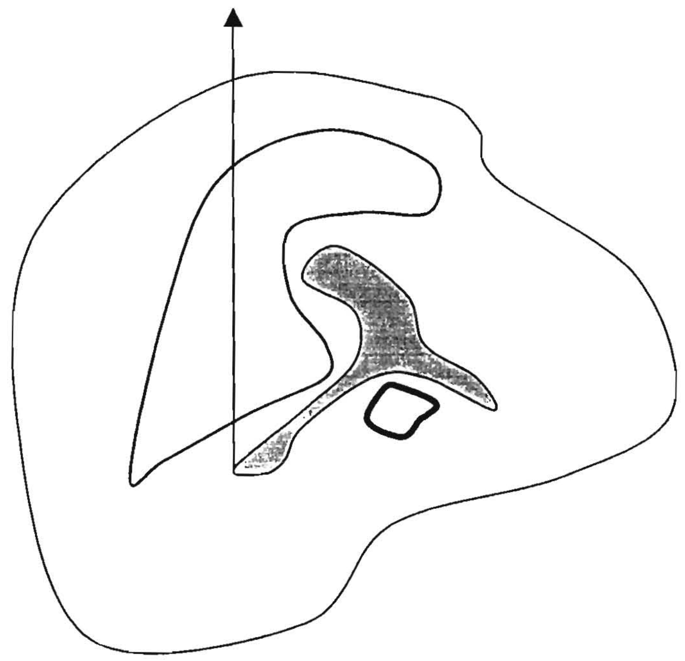
Figure 5.5: Ordering Polygons in Contour File

Rather than describing the detailed process involved, the concept is outlined and these are the main steps:

1. Find the closest true height from the extreme left point of the largest polygon - Figure 5.5, contour 1 (this is the time consuming part of the program). By choosing the polygon with the largest area it ensures that the closest "open polyline" will form the origin for the sequence.

A "Lies Within" function (used to check if a polygon lies within another) is needed. By comparing areas of polygons (bigger areas cannot lie within smaller areas), and extreme points (e.g. if the extreme upper coordinate of contour *X* is less than the extreme lower coordinate of contour *Y*, then contour *X* cannot lie within contour *Y*), a large number of possibilities can be rejected immediately. These operations are relatively quick to perform because the information is available in very small files from step 3.

The remaining possibilities are checked by extending a line from the left extreme point on the smaller polygon parallel to the *Y* axis, and counting the number of times the intersects the larger polygon. If the line intersects an odd number of times, the area lies within the other. Open polylines are ignored in this process. The process for determining intersection is as follows:
Y coordinate of the line is recorded, e.g. *Y* = 100 (the equation of the line);
Origin of the line is also noted, e.g. *X* = 50;
The larger polygon is described by a number of small lines joined consecutively and each is tested for intersection. For all *X* greater than 50, if two consecutive *Y* coordinates lie on either side of 100, then an intersection is noted.



We want to know which area the shaded block lies within.

━━ Not considered – Area smaller than shaded area
── Extended line intersects twice – Does not lie within
─ Extended line intersects odd number of times – Does lie within

Figure 5.6: How the "lies within" relationship is determined

Once an order for the contours has been established, the corresponding heights of the contours are passed to the routine that checks for and corrects errors. After the check has been performed, the data describing these polylines is no longer needed and is removed from the files that stores the data. The largest data file containing the detailed set of coordinates is rapidly reduced in this way. For example, the first iteration alone is likely to check approximately 10% of the data in many cases because it checks the largest polygon (area wise), as well as all polygons lying within this area. On every tenth checking loop, the used data is removed from the data files.

Step 5: Detecting and correcting Errors

The information passed to this process comprises a set of contour height which appear sequentially in the DXF file. The first value is always the true value (closest checked polyline) which is used as the basis for checking the remaining contours. A number of error permutations are likely and separate functions were created to deal with each type of error.

The locations of any errors found are written to a file used to overlay on the new DXF file as a means of identifying which contours have been altered.

Step 6: Writing a new DXF file

It was necessary to create a new DXF file rather than write over the previous file in case some blunder was made. The technique used for writing the file was to copy the original DXF, making the necessary changes along the way. This ensures that the exact format of the original file is preserved, including any abnormalities such as height values being stored in the description field.

5.4.5 Results Achieved

The program was used to check a selected number of the footprints. Most of the errors found in these case are those as shown in Figure 5.4 (right) consisting of snap errors, contour lines crossing, short segments of lines not ending at the border of the region and so on. A smaller proportion of errors consisted of incorrect contour heights which were automatically corrected.

The same program was also used for checking a significant amount of 1:50 000 contour data. A company based in Cape Town (Teramare) used the program to check their data sets (obtained from the SG office) and found it worked on approximately 95% of their files (20Gb). In two cases, the program gave problems due to abnormalities in the data where contour lines represented very steep terrain and a number of contours where missing or crossing each other.

The remaining files were checked correctly and found an average of approximately 40 errors on each file. The program takes in the region of 5 hours to run a 200Mb file on a Pentium 100.

Chapter 6 The 3D Visualisation Program

6.1 Introduction

This chapter is divided in to 3 sections; it:

1. Describes the criterion of the basic VRML scene construction and incorporated into the Internet (Chapter 6.2).
2. Explains some of the advanced technique's use for improving the visual quality of the scene (Chapter 6.3).
3. Describes the custom made tools (programmed using Java) designed specifically for the Laetoli footprints (Chapter 6.4).

Each section is also concerned with performance issues which is a constant consideration when designing a 3D model. Rather than devoting a section to performance, it is discussed throughout the chapter where appropriate.

6.2 Creating a 3D Scene

The building block of a 3D scene in any 3D language are the shapes defining the scene. The design aspects include positioning these objects in a meaningful way. As mentioned in chapter 4, some primitives such as spheres and cones can be defined by referencing the respective predefined shapes supplied by the particular program that is being used. These primitives form part of the language specification which are referenced from a library. More complex shapes are represented by a limited number of polygons which best describe the surface.

The number of points used to describe the scene is the primary indication of the performance that is likely to be achieved. A vast amount of points used to describe an object may result in a very detailed representation, but each additional point requires extra calculations for its display. A greater number of points also means that the VRML file containing this information will be larger resulting in longer time needed for it's Internet transfer. These two considerations (rendering speed and download time) determine the performance. It is an important consideration in design to use an appropriate number of points to maintain a balance between performance and quality.

The problem of deciding on the number of points to be used is an advanced topic and will not be discussed in this section, but will be dealt with in chapter 6.4.8 (Level of Detail). It is more relevant at this stage to outline the way in which a scatter of points is used to create the impression of a solid surface.

6.2.1 Using a DEM to create a Solid Surface

In order to define a solid 3D surface accurately, coordinates on the surface of that object must be known. In the case of the footprints, DEM files for each footprint were available at a regular *X* and *Y* spacing (each point having a corresponding *z* value) of 2.5mm. It is important to note that the points do not have to be arranged in the form of a DEM. Any number and organisation of points describing the surface is sufficient. DEM's were used exclusively in this project for defining the shape of the surface.

DEM files can be used to define a surface in VRML using the following steps:

1. Import the DEM into VRML's file format;
2. Define how the surface is to be drawn by joining nearby points to form polygons, typically triangles. This is a trivial task but nevertheless must be specified;
3. Overlay a surface onto the resulting mesh.

Each of these steps will now be discussed.

Step 1

The result of the first step is shown below. This was achieved by importing the DEM coordinates into a VRML file, and setting the colours of each point as a function of its height.

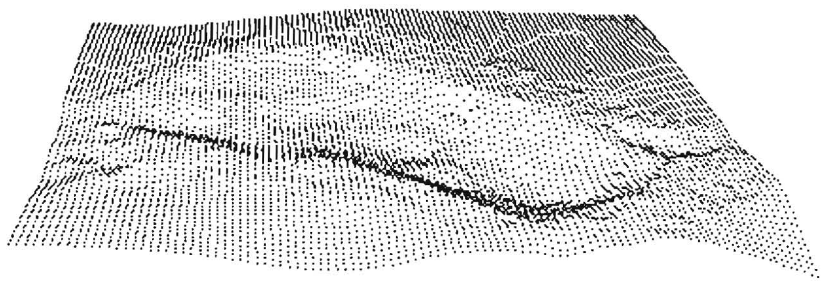


Figure 6.1: Point Cloud

Step 2

If a model is to be displayed as a solid on screen, each pixel value within the border of the object must be assigned a value that best represents the surface at that point. At this stage it is not possible because there is no indication of the terrain lying between points (when using a DEM it is assumed that all points are equally important in defining the topological unit). What is needed is some description of how adjacent points relate to one another so that a shape best fitting the points can be made. In brief, a set of points needs to be used to describe a shape. The way this is achieved is by defining a wireframe mesh which extends the information at explicitly defined coordinates to their surrounding regions.

A wireframe surface is formed by creating closed polygons by means of straight lines passing through points on the surface. Referring to the figure below, the four points (ABCD) could be represented in terms of a surface by joining A-C-D-B-A. By default, polygons joined in a counter clockwise order are visible, otherwise they are transparent. This is a convention followed by most 3D visualisation packages. As an example, the four points below joined in a counter clockwise order would be visible when looking from the perspective shown, but would be invisible when viewed from the other side, since the order they are joined would then appear to be clockwise.

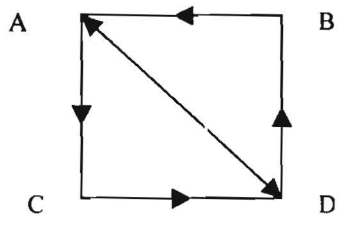


Figure 6.2: Joining Points to Form a Wireframe

Polygons are normally formed as triangles, and so for the same example, two polygons (A-C-D-A, and A-D-B-A) would be specified. This indicates that the region within each triangle forms a plane which is a way of describing the terrain between points. VRML is able to combine the individual planes to form a complete model of the surface.

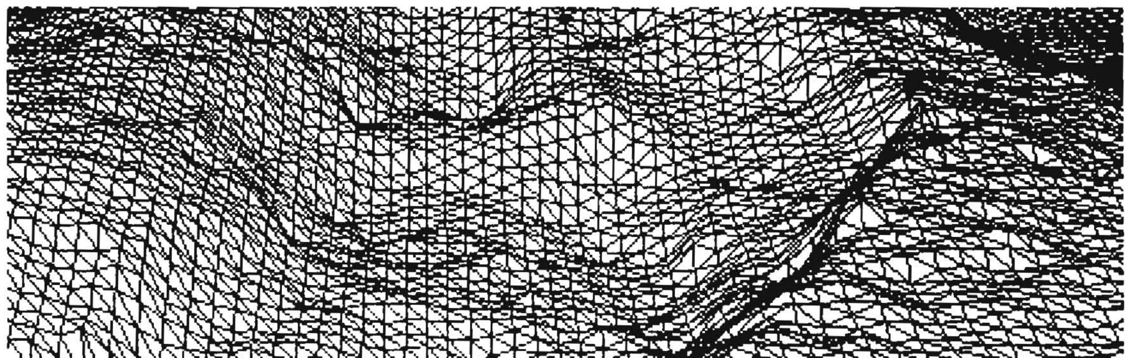


Figure 6.3: Wireframe Grid of a Portion of a Footprint

The information on the structure of the wireframe is conveyed in the VRML file by means of an index of coordinates for each polygon formed. For example, the index for the above example would look like this:

A D B A,

A C D A,

For objects comprising of a small number of points, the process of writing VRML files can be performed manually. In the case of the footprints, there are too many points (and resulting polygons) to write all this information to a VRML file without the use of an automated process. Assuming the region of interest to be 40cm long, 20cm wide, the grid spacing of 2.5mm results in 12800 points (80 rows, 160 columns), and the number of triangles required are then calculated by this simple formula:

$$T = 2\{(r-1)(c-1)\},$$

Equation 6.1: Number of Triangles Needed to Represent a Surface

where T is the number of triangles, r the number of rows, and c the number of columns. In this case, the number of triangles is 25122. A program was required that reads in existing ASCII files containing the DEM values, and produce complete VRML files. The involves (among other things) the sorting of DEM points into VRML format, and the creation of the regular wireframe mesh. This program was written in Visual Basic. This and other programmes are covered in chapter 4.2.

The result created by the Visual Basic program is a relatively large VRML file. A grid of 10 000 points results in a file of approximately 800K. It was apparent that the file size had to be reduced if file download via the Internet was to be practical. A large amount of the data in a file such as this is redundant for the following reasons:

Each *X* and *Y* coordinate is included in the file, whereas it could be specified in a much more efficient manner. Knowing the orientation, origin, number of rows and columns and the *X* and *Y* spacing of a DEM allows the equivalent information to be described at a fraction of the size.

When using a regular grid, specifying each triangle of the wireframe is unnecessary, since there is a regular pattern that is followed. VRML has a standard method for calculating triangles for a surface, provided the height values are ordered correctly, and the surface can be described as $Z = f(X,Y)$, i.e. the surface is 2.5D, which it is in this case.

Applying these principles, allows for a much more compact file to be produced and in this case the files were reduced to 25% of their original size.

Step 3

VRML now has enough information to determine the shape of the object. What is needed at this point is a description of the texture, image or surface that covers the shape. The results of a uniform colour applied to the entire surface is shown in figure 6.4.

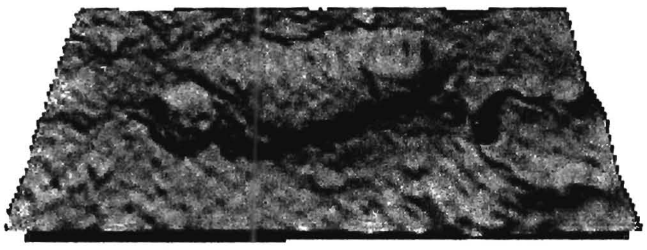


Figure 6.4: Solid Surface

There are numerous settings (e.g. light reflectivity), which are dealt with when refining the scene in it's design stage. For most 3D viewing applications, the result achieved when using default settings is acceptable. The next section deals with some of the refinements that were made to the scene.

6.3 Refining the Scene

Thus far, a set of points describing the surface of an object has been used as a basis for creating a visual representation of that object. The question at this point is, how does a user perceive 3D information represented on a 2D screen? (Brodlie *et al*, 1992) states that the following extra information is essential to perceive the 3D object:

- Context - additional surrounding information can assist in interpretation,
- Perspective - colouring, lighting and shading,
- Movement - having movement is a very good indication of depth, particularly in point scatters.

A number of techniques are available to improve the colouring, lighting, shading and general visual quality of the scene. Other data related to the footprints were also available to be added as a means of increasing the "context" value. The way in which this was achieved and the effect it has on performance will now be discussed.

6.3.1 Colour and Texture Manipulation

The colour and texture of a surface plays a large role in creating a visually appealing or realistic scene. When looking at an object in reality, a great deal of the information that can be gathered relies on the perception of the texture of the surface. One can often perceive an object to be soft or rough, without touching it. We may think of shiny objects as being hard, or semi transparent objects being liquid. The perception of a texture of an object in a 3D scene is created in a similar way, i.e. setting its transparency, shininess and so on. Colours play a major role in humans visual perception (reference), and hence colours can be used to add information to an object. A simplistic example is that a red object is normally thought to be hot and a blue one cold. A 2D image is essentially an arrangement of colours which alone can provide us with a variety of useful information. Choosing the correct colour is an effective tool in improving the quality of visualisation.

Colours and textures are dealt with in the same section, because they are very much interrelated. It is difficult to separate the two because they directly affect each other. A number of fields are used to define colours and textures, each of which will be briefly described. For the purpose of simplicity, in this thesis colour refers to the RGB (red, green, blue) value, and all the other properties fall under the category of textures.

Properties of Colour and Texture Nodes (Hartman & Wernecke, 1996):
Diffuse Colour: Specifies the amount of red, green, and blue that an object reflects from a direct light source. This varies depending on the orientation of the surface with respect to the light source and contributes the most to the appearance of a texture.
Ambient Intensity: Specifies how much ambient light (omni directional) from light sources the object reflects.
Specular Colour: This determines the colour of the light that is reflected directly back to the user off a highly polished surface. This normally applies to glistening surfaces which creates highlights.
Emissive Colour: Specifies light produced by a glowing object.
Shininess: Specifies the reflectivity of a surface, which directly affects diffuse colour.
Transparency: Used to define transparent surfaces.

Effective use of colour and texture is created by use of these fields combined with other more involved settings such as specifying the attenuation (how intensity decreases with distance) of the light and intensity (brightness at the source) of light sources. There are three ways of implementing a colour scheme, each is discussed below.

Single Colour
A single colour can be specified for the entire object. This will result in a very small amount of code since it is not necessary to provide explicit colour values for each point describing the object. VRML provides a node called a "material" node which allows a uniform colour to be used for the entire object. By changing the RGB values of a single field, the colour of the object can be altered very quickly and economically. In some cases this is sufficient, especially if that object happens to have a single uniform colour naturally, or where the shape of the object, rather than its colour is of interest.

The disadvantage is a certain amount of inflexibility because colour of the object must be uniform, and can therefore not be used to highlight specific features of an object. In the field of archaeological visualisation where analysis is performed, detail is important. This means that multiple colours are required to assist in gaining useful visual detail. Using a single colour is an efficient tool, but further colour implementation techniques are required to improve the quality of the scene.

Graduated Colour

Graduating colour according to height aids one in visually determining the shape of the surface. This is shown by comparing figure 6.4 and 6.5. Both are viewed from the same position, and differ only in colour. Graduating the colour in this way can assist the user in interpreting depth in particular. Note how figure 6.5 enhances the deepest areas such as the heel.

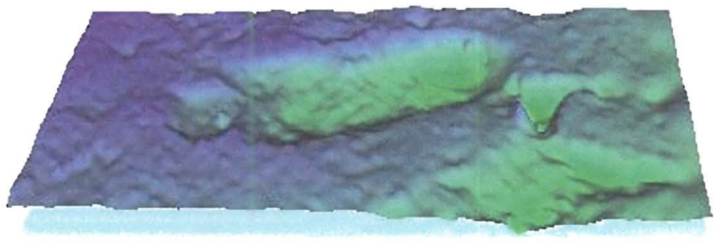


Figure 6.5: Colour graduated according to height

This method is more complicated to implement since colour values have to be given for every vertex of the object. The graduation of colours is then performed linearly between adjacent vertices. This method of colouring is called "colorPerVertex". Not only does this reduce rendering speed, but also results in the amount of code for a file to be approximately doubled. This method has a larger detrimental effect on file size than all the other behaviour techniques combined. In spite of this, it was decided that this function was necessary.

Texture Overlay

It is possible to overlay a 2D image onto an object in a VRML scene. If the image represents the same area as the 3D data with no distortions, combining the two provides the best way of representing the original scene. Such an image was available for this project and is created by using the DEM and therefore match the model identically.

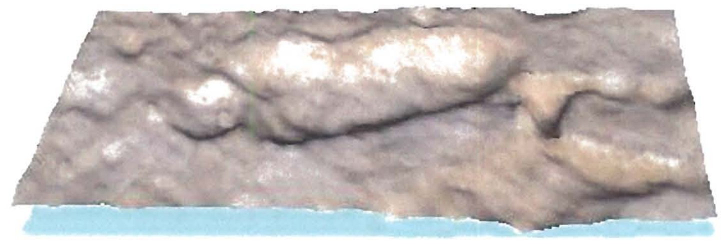


Figure 6.6: Ortho-Image Overlay

The ortho-image has information which enables a more realistic view of the footprint to be created. When the ortho-image is overlaid, the geometry of the footprint is complemented by the information available with traditional means of data representation in terms of an ortho-image. Although an ortho-image is geometrically correct, it is difficult to perceive height variations in a 2D image. Conversely, 3D data is good for showing the shapes of objects, but without 2D overlays, lacks important information regarding the colour and texture of a surface. When combining the two data types, a realistic scene showing both the topography and original texture of the object is created.

The performance in terms of rendering speed is largely determined by the size of the image that is overlaid on the surface. A once off download is required to acquire the image over the network. If the detail of the image is kept low enough, the rendering performance can outperform that of the graduated colour method.

6.3.2 Smoothing the Surface

Once a surface has been created, some techniques are used to make the shape of the object seem more realistic. In theory, a true representation of a non regular shape could only be made if an infinite number of points were used to define it. Since this is not possible, and planes are used to interpolate values between points, it is possible that a surface that is curved (or smooth) in reality appears to have jagged edges. This jagged impression is created on screen because the wireframe described above creates a set of planes which link together at angles which makes light reflect differently.

The impression of a smooth surface is created by reducing the extreme changes in direction of light reflectivity at these junctions. Each pixel value is calculated by amongst other things, the way light reflects off the surface at that point, and if smooth transitions are to be made, it would require small variations in the direction of which light reflects between neighbouring pixels.

The way in which light reflects off the surface of an object is determined by the normal vectors to the surface at each point. Two methods are used to specify normals, namely "Normals per vertex" and "Normals per face". The Diagram below shows the differences in appearance resulting from the two methods. (Hartman & Wernecke, 1996)

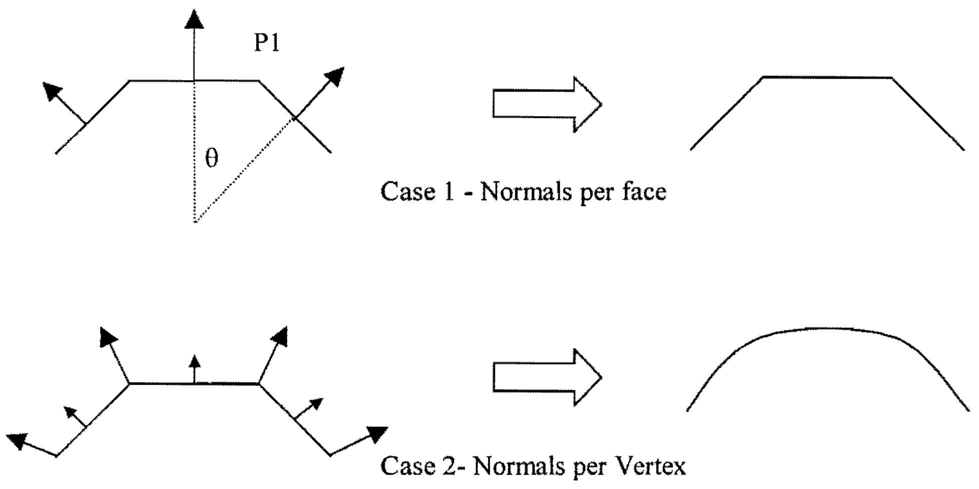


Figure 6.7: Methods for Specifying Normals

It is apparent from the figures above that for the creation of a smooth surface, "Normals per Vertex" is the better option. As their names suggest, "Normals per Face" calculates the normals to the surface from the plane defined by the lines joining adjacent points. "Normals per Vertex" calculates additional normals at each vertex by averaging the values of the normals of the faces at each side of a particular point resulting in smaller variations of the direction of reflectivity at adjacent faces and a smoother appearance.

Sometimes there is a need for a sharp edge to be shown as such. As an example, a corner of a wall may naturally have a sharp defining edge, and may be curved in other parts, where it needs to be smoothed. The "creaseAngle" is used to determine which edges are to be smoothed, and which ones should have a sharp edge. The creaseAngle is a positive angle defined at design time which is compared to the normal angles between adjacent edges. From Figure 6.7, the angle θ is the normal angle at point P1.

If the normal angle is smaller than the creaseAngle then the edges are smoothed, otherwise they are faceted. The effects of this method are illustrated below.

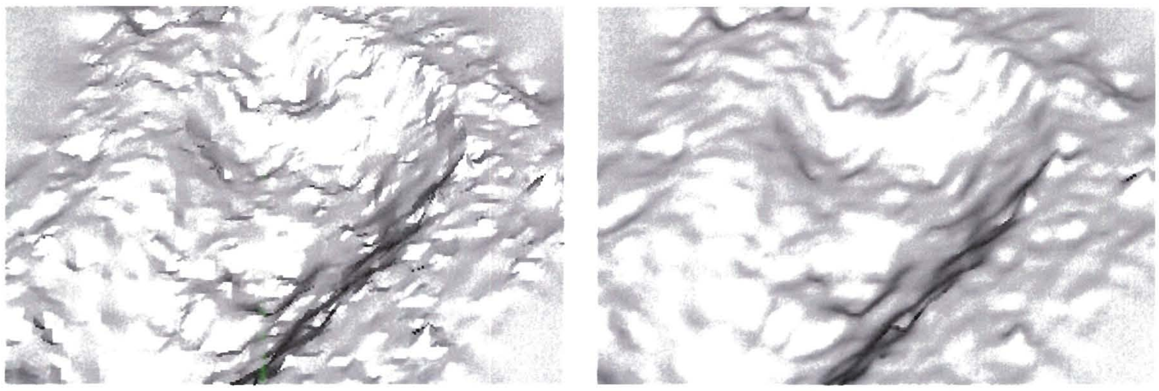


Figure 6.8: Using "creaseAngles" to Smooth the Surface

These two techniques are the most effective way of creating the impression of a detailed image without increasing the number of polygons. Edges can still maintain a sharp resolution and are not blurred, and the resulting scene gives the impression of containing more information.

6.3.3 Viewpoints

Viewpoints are a means of regaining orientation when the user becomes disoriented. Positions and orientations of viewpoints are specified to which the user can "jump". This is very useful for 3D visualisation of architectural structures where the user is more likely to get disoriented. Viewpoints may be located in each room enabling the user to be teleported to a specific room without having to navigate through other rooms and passages. In the case of the footprints, three oblique viewpoints, and one top viewpoint were specified.

6.3.4 Multiple objects

Thus far, only single objects in a model have been discussed. Multiple objects in a single scene are likely to result in larger files, but have the advantage of making comparison of images easier.

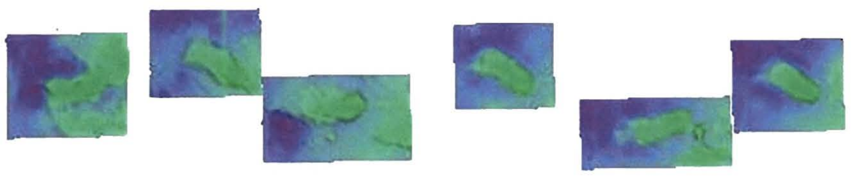


Figure 6.9: Multiple Objects in a Single View

The figure above shows a set of footprints which create an impression of part of the trackway. Combining objects is done by ensuring their relative positions and orientations are maintained. This was done in Visual Basic by creating a project which writes individual files of each footprint, while keeping a record of their relative positions.

Certain techniques are used to compensate for the loss of rendering performance. The main method for compensating for the increased polygon count is to use different levels of detail, depending on the distance the user is from each object. This is discussed in more detail in chapter 6.4.8. The scene shown above has four levels of detail which change automatically when the user reaches a certain distance from each footprint. The distance at which the level of detail changes must be carefully chosen in order to give the impression that the original scene has not been altered. This is discussed in section 6.4.8.

6.4 Implementing Behaviour Using Java

The following section deals with all the functionality that was added to the project using JAVA. As stated earlier, behaviour includes any functionality that would not be expected of a standard 3D scene. A standard 3D scene must be expected to be viewed from an infinite number of viewpoints and thus have a means of rotating, panning and zooming the object so these functions are not classed as behaviour. Clicking a button that changes the colour of an object within the scene would not be a standard operation, and is therefore classed as behaviour.

Determining which behaviour is to be introduced is largely dependant on the type of application. For example, architectural applications require easily navigable scenes where the user can regain orientation easily, rather than a complex set of measuring tools. Creating a realistic looking world may be achieved by paying attention to lighting of the rooms rather then very accurate coordinates of the data that is used for creating the model of the structure. This means that this type of application does not require much in the way of behaviour.

The set of tools described here was designed specifically for the Laetoli project but can nevertheless be used in other applications in certain cases. Some ideas of these possible uses are briefly mentioned.

Each tool is described using the following format (where appropriate):

- What the tool is meant to achieve,
- The method, algorithm, or formulae used for programming the function,
- Possible alternatives or shortcomings of the tool,
- The resulting performance achieved when using the tool,

- Possible applications of the tool for use in the Laetoli project and other applications.

6.4.1 Height exaggeration

As the name suggests, this is a tool that exaggerates the height scale, while keeping the horizontal scale uniform and is achieved by multiplying the height values by a factor in the range of 0.1 to 3.0. It is a useful aid for increasing the effects of shadows and for determining peaks and valleys as well as magnifying small changes in height that would otherwise not be noticed.

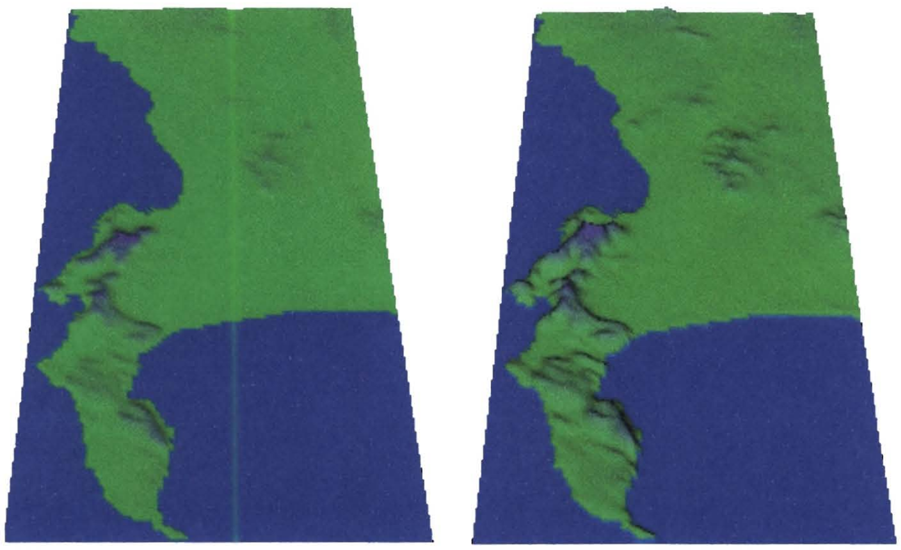


Figure 6.10: Cape Peninsula

Changing scale is more useful in situations where the natural topography is relatively flat. This is very noticeable when dealing with topography of very large areas, where a surface model tends to create the impression of a very large flat area. Consider figure 6.10 showing a VRML of the Cape peninsula and surrounding regions. In order to make the result appear undulating, a height exaggeration of 1:3 was used to accentuate the mountainous regions. Even with the heights being stretched as they are, without the use of colour to separate the sea from the land, and darker colours representing higher terrain, height variation is still difficult to be shown.

Perhaps the main benefit in terms of the footprints, is modifying the apparent depths of different prints to make them coincide. It seems that the ground at certain places along the track is softer than other parts which leads to some footprints being deeper than others. In order to compare footprints that have different depths, it may be useful to exaggerate the scale of one of the prints in order to resemble their appearance on the same ground. Assuming of course that whoever was responsible for making the footprints did not become any heavier while they were walking – it can be assumed that the footprints should be the same depth.

Similar analysis could be performed by manipulating the height values in order to make the footprint correspond to imprints in different surfaces. A human footprint could be made in sand for example and used for comparison. The depths of the imprint are unlikely to be the same due to the different surfaces, but by modifying the height scale of either of the footprints, a better comparison could be made.

Performance is not affected by using this method because a once off calculation altering the heights is performed, and the resulting model is no more complex than the original.

6.4.2 Flood Plane

The flood plane is a horizontal plane which can be moved through the surface of the footprint and represents water at a given level. The flood plane is transparent enough to see the detail that it covers. Examples of the possible uses of the flood plane:

- The outline represents a contour at any required height.
- Makes it easy to identify the deepest area by moving the plane until the water is just visible. Similarly, for the highest point, the flood plane can be moved until only the highest area is not submerged.

By moving the plane a little below the approximate average ground level, the shape of the footprint can be seen.

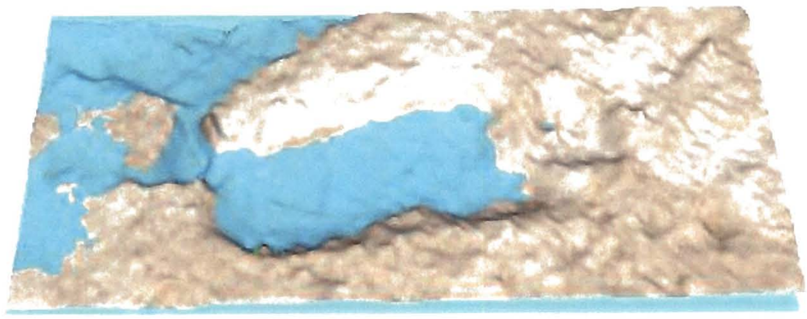


Figure 6.11: Flood Plane

6.4.3 Gradient Highlighting

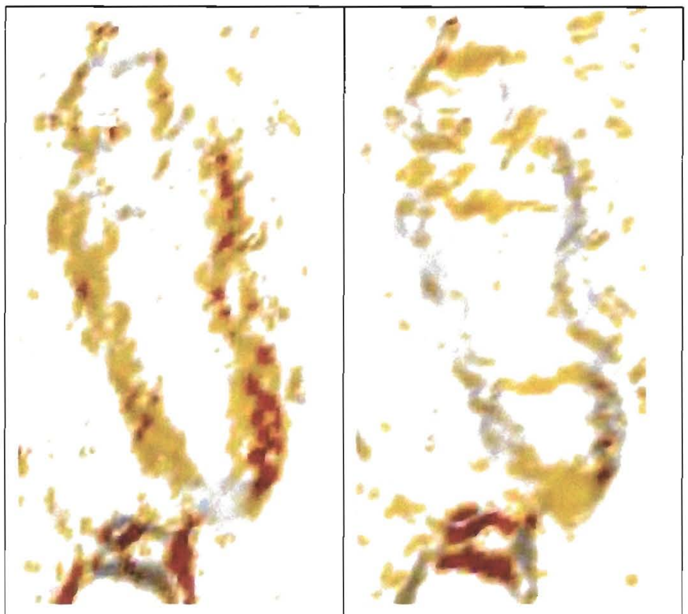

Case 1: East/West   Case 2: North/South

Figure 6.12: Gradient Highlighting

Gradient highlighting is used to represent different gradient with different colours, ranging from red for a steep gradient, to white for a level gradient. The gradients can be calculated along four directions (N-S, W-E, NW-SE,NE-SW).

This is calculated by using a regression line through three points. In the illustration below the gradient at P1 is calculated by using the two neighbouring points, and determining the line that minimises the sum of the squares of the vertical distances from the points to the line.

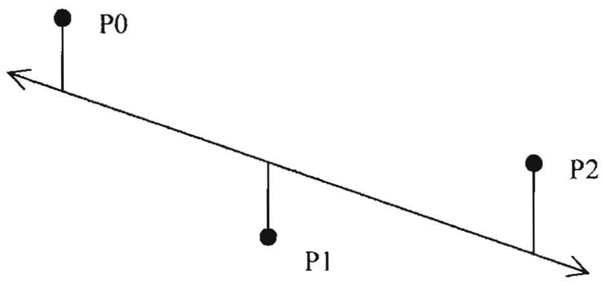


Figure 6.13: Regression Line

$$m = \frac{\left(\sum x_k\right)\left(\sum y_k\right) - n\sum x_k y_k}{\left(\sum x_k\right)^2 - n\sum x_k^2}$$

Equation 6.1: Calculating the gradient of a regression line (Thomas & Finney, 1992)

This calculation is performed for all points describing the footprint (using equation 6.1) and then a colour is assigned to that point depending on the magnitude of the gradient. To illustrate the use of gradient highlighting, consider figure 6.12. These two images are taken from the same position, the only difference is the direction in which the gradient was calculated. Case 1 highlights the side walls of the footprint whereas case 2 highlights the base of the heel and the end of the toe. The maximum (or minimum) gradient can also be shown which uses the maximum (or minimum) gradient of the four possibilities to calculate the colour for each pixel.

6.4.4 Distance Measurement

The distance between any two points on the surface can be calculated as a 2D and 3D distance measured to the nearest millimetre. For this XYZ coordinates are required and are obtained from the VRML file at run time.

The difficulty involved for all the measuring tools was largely due to problems associated with extracting coordinates of a mouse click on the object. This is a technical problem of both extracting screen coordinates, and relating them to the coordinates of an object within a scene. This is not a difficulty with many other packages which provide ways of easily accessing the mouse coordinates. As an example, when using Visual Basic, screen coordinates of the mouse are tracked continuously (regardless of whether the mouse is over a button, a picture etc.) and are set as global variables *X,Y* without the need for any programming. VRML does not offer an equivalent function, but it does allow object coordinates to be returned, if the object pointed to is a *touchSensor* - the same mechanism used for creating light switches that can be clicked to trigger light sources. Coordinates obtained from a *touchSensor* are as accurate as the pointing of the mouse, i.e. they are accurate to a single pixel value. This implies that greater accuracy is obtained by zooming closer to the point to be selected when choosing it. This is performed at both terminals of the measuring line , and the resulting distance is returned to the nearest millimetre. To give a rough idea of the sizes of the footprints, a 2D distance of G1-25 from the back of the heel, to the end of the big toe gives 22.4cm.

The chief shortcoming regarding distance measurement is that distances between different footprints cannot be calculated. The principle used so far could easily be extended to achieve this, but since only a single print is used in the analysis, it is not possible to relate information to adjacent footprints. It would be easy in theory to do this but the problem is that a number of footprints in a single model would result in very slow rendering.

A possible solution to this problem would be to enable the user to choose a second print to be included in the model, viewing both at a lower level of detail. It was decided that tools for analysis would be focused on single footprints and so this feature was not included.

6.4.5 Cross Sections

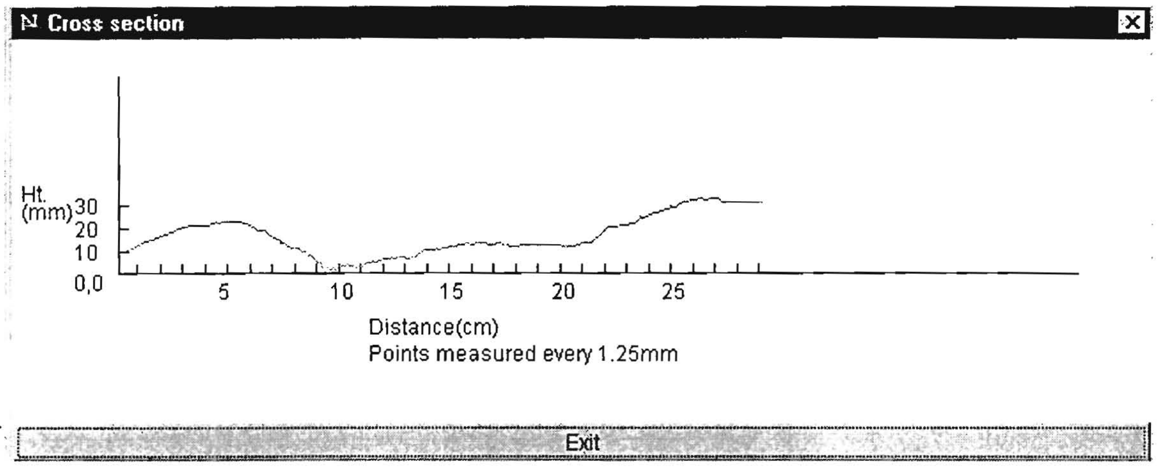


Figure 6.14: Cross Section at a 1:1 Scale

The cross section provides a way of relating heights and distances along a given path. In this case, points taken every 1.25mm, create a profile between any 2 points at a 1:1 scale. The cross section obtains information of the prints along a line joining two points. Providing the user with a means of extracting useful numeric information is the task of the designer, and without specially written procedures, there is no access to any numeric data. Applications such as cross sections show the potential of the types of behaviour that can be programmed. The cross section extracts data already available in the 3D file in the form of DEM points, and rearranges it into another form of display useful to the user.

Profiles are a means of comparing the shape of the Laetoli footprint with the equivalent section of a "modern" footprint. Although casts of the footprints have been made, it is difficult and time consuming to manually determine cross sections accurately. This type of application is a prime example of the benefits of digital data compared to existing forms of data. Any number of cross sections can be created almost instantly whereas previous methods of gathering this type of information where laborious and time consuming.

Method used for calculating the cross section:

Step 1:

The start and end points (terminals) of the cross section are obtained from the values used for the distance measurement procedures.

Step 2:

These two points are then used to calculate:

The number of points needed to calculate the cross section using a spacing of 1.25mm,

The direction between terminals, and thus the 2D path of the section.

Step 3:

The x and y coordinates of each point in the cross section are then calculated by selecting points along the 2D path at a regular (1.25mm) interval. This enables certain parts of the cross section diagram to be calculated and drawn (such as labelling) while the height values are being derived so as to save time.

Step 4:

The heights of points along the section are then calculated. The x and y coordinates are used to determine which block of the grid DEM each point lies in.

Since the grid is square, the vertices of the square are the four closest points to the point in question.

Rejecting the furthest vertex (from the point in question) will also mean that the point will lie within the remaining 3 vertices. This enables the height to be calculated by assuming that the 3 vertices form a plane, and then working out the corresponding height for the x,y in the plane.

A possible weakness of this approach is that a linear method is used to calculate the heights along the profile. An alternative approach would be to use all four vertices surrounding a particular point in the calculation of the height value. It was decided that the trade off in lower accuracy for increased speed was worthwhile, considering the very small loss of accuracy (should never exceed 0.5mm; example given below). The heights along the profile are calculated by using formula 6.1. The alternative approach would be to use equation 6.2.

Using the closest three points to calculate height:

$Z = a_0 + a_1 X + a_2 Y$

Where the coefficients a_i are given by:

$$\begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & X_2 & Y_2 \\ 1 & X_3 & Y_3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

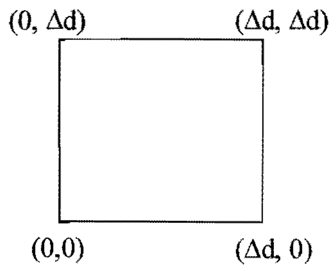
Equation 6.2: Find the Height of a Point Lying in a Plane

Using the closest four points of a square to calculate the height (Kraus, 1996)

$Z = a_0 + a_1X + a_2Y + a_3XY$

Where the coefficients a_i are given by:

$$\begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \Delta d & 0 & 0 \\ 1 & 0 & \Delta d & 0 \\ 1 & \Delta d & \Delta d & \Delta d^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$



Equation 6.3: Finding the Height of a Point Lying within the Bounds of a Square

Consider the following example:

Here A, B, C and D are points which have known (X,Y) coordinates, and we wish to find the corresponding height. Using the three closest points, equation 6.2 is used to find the formula for the plane, and using the closest four points, equation 6.3 is used.

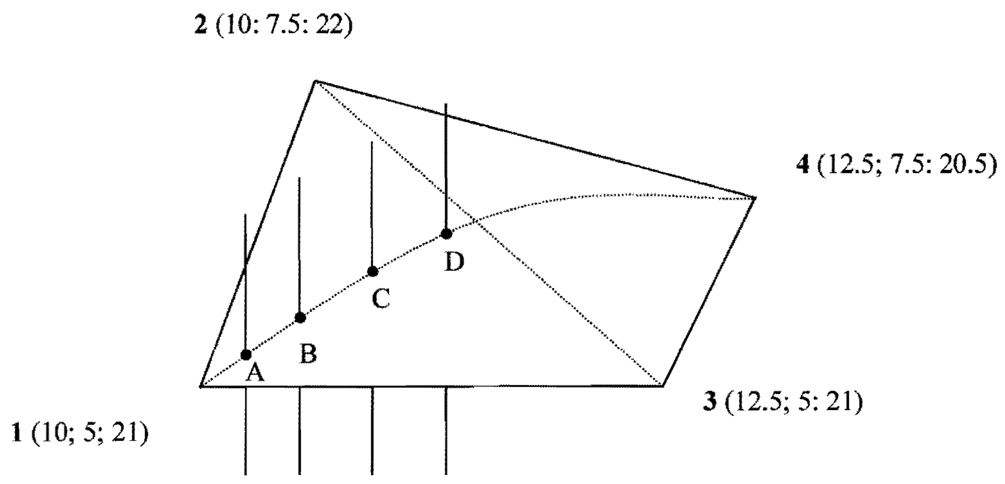


Figure 6.15: Finding Z for a Given X,Y

The two different methods achieve the following results:

Table 6.1: Heights Obtained from Interpolation

XYZ (mm)	Height (mm) (Using Equation 6.2)	Height (mm) (Using Equation 6.3)	ΔH (mm)
A (10.25, 5.25)	1.1	1.085	-0.015
B (10.5, 5.5)	1.2	1.14	-0.06
C (10.75, 5.75)	1.3	1.165	-0.135
D (11, 6)	1.4	1.16	-0.24

For points close to one of the corners, the difference is small, and increases for points closer to the centre of the square. In this example the maximum difference between these two methods is 0.24 units. In the case of the footprints, this corresponds to 0.24mm. It was decided that using three points was accurate enough for the task of producing a cross section.

The result of the cross section are shown in the figure 6.14. A typical cross section may be 35cm long (280 points) which results in an insignificant number of numeric calculations. The time consuming factor for this process is the transfer of data between VRML and Java. A cross section takes approximately 2 seconds to calculate and draw (using a Pentium 233).

6.4.6 Flythrough

A flythrough is created by automatically transporting the user through the scene via predefined viewpoints. This is especially useful for introducing a non experienced user to the scene showing a moving scene that does not have to be controlled manually. Inexperienced users may find it difficult to move around the scene and the flythrough is a means showing the type of movement that can be performed.

Method for creating a flythrough:

This is achieved by using a number of sensors that trigger one another in such a way as to maintain a uniform speed of movement irrespective of the hardware used. The process uses the following functions which have already been discussed in the previous section:

Proximity sensor: Detects when a predefined region in 3D space has been entered by the user.

Time sensor: Can detect when this region was entered, and the period that has elapsed since.

Orientation/Translation Interpolators: An arbitrary (but equal), number of both must be defined at design time which specify set locations in the scene. Linear intervals between adjacent locations can then be interpolated. The number of intervals is controlled by the time sensor which must complete the whole process in a given time.

Detailed approach.

Step 1:

Bind the user to the "top" viewpoint. This means that the user is fixed to that viewpoint, and any changes that occur to "top" will automatically transfer the user. The "top" viewpoint is used because the flythrough terminates at the position of that viewpoint, and thus when the process is complete, the viewpoint can be "unbound", setting the scene back to normal.

Step 2:

Moves the user to a proximity sensor (with a very small radius) which detects that the user has entered the region and begins the flythrough. A flythrough is achieved by the use of a principle called "keyframe animation" which was mentioned in chapter 4.5.4. One would expect animation to be implemented on an object to create movement, but it can also be applied in other ways to create various effects. In this case, the viewpoint of the user passes through an animated sequence creating the impression of motion.

Consider the following example where code (roughly resembling that in a VRML file) is used to explain how "keyframe animation" was applied to the flythrough:

Key [0; 0.3; 1] - comprises of two or more values, the first must be zero and the last one.

Value [x_1 y_1 z_1; x_2 y_2 z_2; x_3 y_3 z_3] - Contains a coordinate for each "key".

Time interval 10 - indicates that the process must reach completion in 10 seconds.

The key defines key moments in the animation sequence calculated as a fraction of the time interval. Here, three moments are calculated as a percentage of the time interval of 10 seconds, i.e. the first moment is at 0 seconds, the seconds at 3 seconds, and the third at 10 seconds. Each "Value" corresponds to a "key", so at time 0, the viewpoint will be at position x_1 x_2 x_3, and after 3 seconds the viewpoint will be at position x_2 y_2 z_2. The browser will calculate the necessary frames that are to be rendered in getting from one keyframe to the next by means of a basic linear interpolation. The number of frames to be interpolated will depend on the rendering speed because the animation specifies that after 3 seconds, the viewpoint must be at a certain position which may typically imply 30 frames (assuming a frame rate of 10Hz).

Step 3:

The results achieved from step 2 are a set of viewpoints along the path which are adopted by the "bound" viewpoint which then carries the user along with it. The flythrough ends at the position that the "top" viewpoint is set at, and then unbinds itself and the movement controls are back to normal.

One of the main difficulties regarding viewpoints, is the selection of the translation and orientation values (key moments and key positions). Not only are those values different for each footprint, but obtaining effective values is often a matter of trial and error. In the case of the footprints, the desired path for a flythrough is difficult to design because there are no obvious points of interest. Certain applications do have more predictable points of interest, for example a view from a car window in a virtual world would, with a little luck, follow the road. In either case, the flythrough has the weakness that the path cannot be changed by the user. A predefined path is set, and if this is not suitable then the standard movement tools must be used.

For multiple footprints, a flythrough set at approximately 1.5m above the surface, looking down can create an impression of what it would look like walking over the footprints. As mentioned previously (section 6.3.3 – viewpoints), this type of application is more suited to architectural type environments where movement is more restricted by obstacles such as walls and doors, and a predefined path through such passages is likely to make movement much easier, especially when the path that the user wants to travel can be predicted.

6.4.7 Contour Files

Traditionally, contours are used to supplement 2D data by providing visual information of the height variation in topography by joining points of equal height. Combining contours with the DEM as shown supplements the 3D data especially for those who are accustomed to viewing contour files and are aware of their usefulness. Contours reveal subtle surface variations that are not immediately apparent to the eye when viewing an image (Greene, 1983). Combining data in such a way is important in increasing context value – it is often the case that viewing two objects together reveals more than viewing them separately.

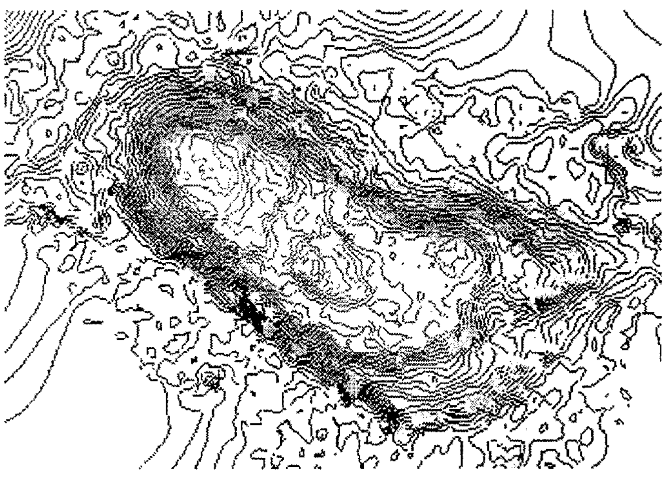


Figure 6.16: Contour Overlay

When using contours in 3D visualisation, it is possible to draw each line at a different height, thus reducing the need for explicitly giving their height. Relative height difference between contours can be deduced visually. Leaving out height values however means that absolute heights cannot be deduced. The contours are given colour in the same way as for the graduated colour of the footprint. They can be viewed independently, resulting in very much improved rendering performance (line drawings render very efficiently, as there is no need for shading calculations), or overlaid on the surface as shown above.

Method for Creating VRML Contour Files:
The contour data was available in DXF format. A number of programs were written to convert these files into the required VRML format.
A checking routine to check the contour files for errors was written in JAVA (Chapter 5.4);
A second JAVA program was used read DXF files and outputs XYZ files. Included in this was a routine to reduce the vast number of points to a number practical for use in VRML (Chapter 5.2.1);
The XYZ file was then converted into a VRML file using Visual Basic.

Since the contour data is written independently to the footprint files, they can be viewed independently and when needed are called from a separate file. This means that when the project is loaded, the contour files are not transferred, and thus do no slow the process. If the contours are requested by the user, they are transferred. By being stored in the users cache for further use they do not need to be transferred again.

Possible improvements to the contour data would be to include textual data to give the absolute heights of contours. This would however clutter the scene, and absolute heights are less useful when working a single object because values cannot be compared to other objects of interest. When dealing with a single object, the relative heights are of interest.

6.4.8 Level of Detail

Level of detail (LOD) is used to increase rendering speed by switching to a lower detailed representation of an object. A less detailed image or object normally replaces a more detailed one when the distance from the viewer makes switching undistinguishable to the human eye.

The technique of switching between levels is normally programmed to occur automatically at a particular distance (this was done for the trackway). For the individual footprints, the switching is performed manually by the user, and therefore the difference in appearance is expected to be noticeable. This is done for the purpose of benefiting a user who is either satisfied with viewing a lower detailed scene with the resulting increase of rendering speed, or has limited hardware to deal with the large amounts of processing required for larger scenes.

A lower level of detail can be used when position and orientation of the footprint is changed or a flythrough is performed. The result is faster and smoother rendering, and since no analysis is likely to be done while the image is moving, the lower detail is sufficient. For the purposes of analysis (such as cross sections), the higher level of detail can then be applied.

Method used for Creating Levels of Detail.
VRML has an effective method for switching between level of detail if the switching criteria are defined at design time. In this case though, the user is expected to change the level of detail which is not a standard function of VRML. A way around the restrictions imposed on the LOD node had to be found. Rather than attempting to explain the technical details, this section will briefly explain the basic concept and the resulting performance achieved.

In brief, PROTOTYPING (chapter 4.5.3) was used to create two skeletons for each footprint. The one at the higher level of detail is the default print which is used until the user specifies otherwise. When the other PROTOTYPE is chosen, all changes made to the original footprint are noted and these changes are then applied to the skeleton of the other level of detail, and vice versa.

Using PROTOTYPES for this application has the following benefits:

- The files become much more manageable. The data defining the shape of the footprint (which remains static) is separated from the rest of the VRML file (which often needs to be edited and modified at run time).

- It provides some form of security since PROTOTYPES do not have the same format as a typical VRML file, where a coordinate list is given. Changes in naming and ordering of data make it more difficult to copy.
- It improves performance because the browser has an indication of what can and cannot be changed, and thus is able to optimise the parts that are fixed. When defining a prototype, one must indicate which parts of the file may be modified, and the rest is assumed to be fixed, reducing the number of possible alterations that an be made to the scene.

The following table shows the frame rates obtained using the two levels of detail using G1-25. The results will differ vastly depending on the hardware used. The contour files are not shown in less detail, but are included to give an indication of the influence of line rendering.

Table 6.2: Frame Rates with Various Overlays

G1-25, South Track	18304 Polygons (Frames per Second)	4576 Polygons (Frames per Second)	Percentage Improvement
Contours Only	7.0	7.0	-
Plain Texture	3.5	7.9	125
Ortho-image	2.8	6.1	117
Plain Texture + Contours	2.7	4.3	59
Ortho-image + Contours	2.2	3.6	64

The number of polygons to be drawn can be calculated from equation 6.1. The lower level of detail is created by using every second point in each row, and every second point in each column. Thus, the number of polygons for the lower level of detail is approximately a quarter that of the high level of detail.

6.4.8 Light Manipulation

Lighting is one of the most mathematically involved technical issues of 3D visualisation (the equations used for describing different light sources will not be dealt with). In basic terms, the intensity of light at a point combined with its RGB value (as specified in the design stage) form a colour that is seen on the screen. Light intensity ranges from 0 to 1 and this factor multiplied by the RGB values at a point gives the resulting colour. An intensity of zero results in no RGB (black), and an intensity of 1 will result in the full RGB colour to be shown. Some 3D packages allow for negative intensity values (light sinks) but VRML does not have this function. The three basic methods for illuminating a scene in VRML are briefly discussed.

Point Light

Illuminates equally in all direction from a particular point in 3D space. Various properties can be set in order to describe how far the light reaches and how the light attenuates. Linear attenuation of light is the most basic approximation which allows for more simple lighting calculations. The most realistic result is obtained by using quadratic attenuation (Hartman & Wernecke, 1996) which unfortunately has a detrimental affect on performance. A point light is ideal for modelling a glowing object, where the light appears to originate from a single point.

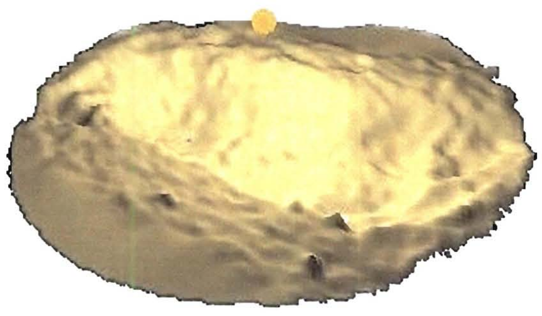


Figure 6.17: The Effects of a Point Light Source

Spot light

As the name suggests, this light behaves similarly to a torch or spotlight. The light originates from a point and illuminates in a given direction with a given cut off angle. Thus the portion of the 3D scene that is illuminated will take the shape of a cone, where the vertex of the cone is the specified location of the origin of the light.

Directional light

Directional light illuminates along rays parallel to a given 3D vector and is the standard form of lighting because it is best suited for uniform illumination of a scene. In VRML the standard light source is referred to as the headlight. A directional light source is used to model the sun, where the light rays appears to be parallel as if arriving from a source infinitely far away. A point light behaves more and more like a directional light, the further the point light is from the object. It is possible to use the point light as the sun if the point light is far enough away from the object that it illuminates.

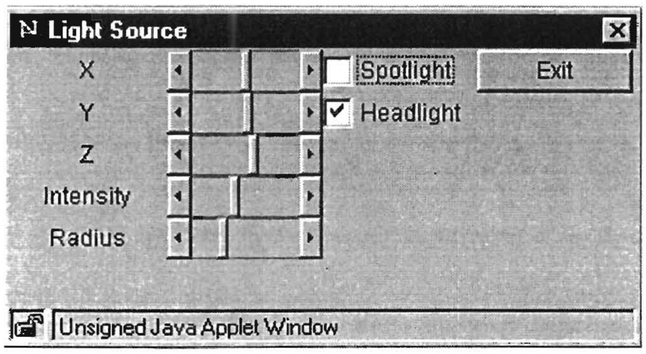


Figure 6.18: Java Applet for Changing Light Settings

By switching the default light (directional light) off, and using a point light, shadows can be created from different angles because the object can remain fixed while the light is moved over the surface. The radius and intensity of the lights effect can be set so as to illuminate a particular region of interest, as in the figure shown above. In this case, a bright point light was placed above the footprint and the radius set so as to highlight only the footprint and not the surrounding areas. When rotating the footprint, the point light will move with it. This has the advantage of keeping the shadow effects fixed. This does not happen when using directional light, and the changing shadows can effect the interpretation of colour and texture.

Altering light settings allows more interaction with the scene which may improve the aesthetic appeal of the object. This particular tool does not slow rendering down in any way, and the code for this function is negligible in size in comparison to the rest of the project.

Chapter 7 Conclusions & Recommendations

7.1 Conclusion

Since the inception of the Internet, the scientific community has been quick to exploit it's potential by presenting data with animations and video clips. Certain data is represented more effectively in this manner rather than in professional publications which are comparatively expensive to produce. However these are still static forms of data representation in which the distributor assembles the data which depicts a scene that has been witnessed. The result cannot be explored in a way that it would be in reality but is biased by way the result is presented. 3D visualisation provides a partial solution to this problem because an accurate model is provided which can be viewed from any desired position in order to gain an unbiased personal impression. 3D visualisation has been used successfully in standalone applications in fields such as medicine, architecture and engineering both as a viewing tool and for analysis. This thesis used 3D visualisation on the Internet to present archaeological data in a way that promotes interaction and analysis.

At the outset, question were asked about the practicality of 3D visualisation. The thesis went on to explain how 3D visualisation was used and combined with the Internet to allow for active participation with the data. In doing so, answers to these questions were obtained and are now summarised.

7.1.1 Concluding Analysis

To what extent is 3D visualisation restricted by using conventional hardware?

The degree to which hardware restricts 3D visualisation is largely dependant on the expectations of the user. Movies using 3D visualisation to generate special effects have created unreasonable expectations for many. Photo realistic impressions, advanced animation and frame rates used in movies are not yet common place using standard PC's. Assuming this standard to be the ideal goal, hardware does impose large limitations.

To create a perfectly accurate description of a model, an infinite number of points are needed to describe it's shape. A computer has limited storage space and more importantly limited processing power. A compromise is needed in using a sufficient number of points to approximate the shape while operating within the confines of the hardware capabilities. In order to maintain an adequate frame rate, a limit of 15 000 points is suggested (using a 1999 standard PC). This restriction can lead to disappointing results especially in cases where large areas are modelled or very complex shapes are described. Detail on the model is sometimes lost and undulating surfaces become jagged.

A further consideration is that of describing colour and texture of a surface. Ideally, one would require an ortho-image of high resolution to depict a scene satisfactorily. Computer restrictions make this impractical though and further compromises need to be made. When appropriate, single uniform colours and lower resolution images are used.

The effect that these restrictions have varies from model to model. A general tendency is that man made objects are less effected than natural objects. Although it is not always the case, consider the difficulty involved in modelling a natural object such as a tree or even a piece of undulating terrain. The surface variations cannot be represented easily without a vast number of points being used. The opposite normally applies to man made objects such a buildings which can often be represented by primitives such as planes, cylinders cubes and so on.

Hardware does impose restrictions on 3D visualisation. Some disciplines such as architecture are affected very little by the restrictions, and other disciplines are severely constrained. Some complex shapes can be adequately represented such as those surveyed using close range techniques where the detail can be maintained using only 15 000 points. Large scale areas are difficult to model (when they cannot be described by using primitive shapes) and will continue to be so even when advances in computer hardware lessen these restrictions. The same principles will apply making certain 3D reconstruction very difficult. In cases suited to 3D modelling, the results of both appearance and performance will improve with hardware advances.

Does 3D visualisation provide a practical means for analysing data?

For applications suited to display in a 3D environment, the possibilities for analysis are very encouraging. Close range photogrammetric techniques in particular provides data ideally suited for analysis in a virtual environment. The suggested limit of 15 000 points is not likely to result in a poor representation of shape and the ortho-image can be used to identify points of particular interest when overlaid on the model, even if the resolution is reduced to maintain an adequate frame rate. Of course, models do not only represent things that already exist, they can also be used to represent proposed design plans for example. A proposed design for a construction such as a house can be tested giving a more realistic feel than sketches.

Viewing 3D data in itself has benefits for analysis. The freedom to view an object from any desired angle and distance, as well as the more descriptive representation makes 3D visualisation more valuable as a presentation tool than photographs or sketches. The real power of 3D visualisation becomes clear when the functions used to manipulate the model and extract data from it are appreciated. This aspect is not limited by computer technology. Any conceivable tool could be programmed to assist the user. The model can be deformed, the colour changed, and data extracted and converted into any form desired. The results appear almost instantly, there is seldom a frustrating waiting period while results are processed.

Many applications are well suited for analysis in a 3D environment. The following are suggestions of what could make 3D visualisation impractical as an analysis tool:

1. The required analysis can be performed easily using existing methods. 3D visualisation is simply not a practical tool for all tasks.
2. The model cannot be represented adequately in 3D. This could be due to the hardware restrictions mentioned above or the difficulty involved in the data collection process. Some data collection procedures require expensive equipment which may be beyond the means of some users.
3. The manpower required for programming the specialised tools is not available. This is the limiting factor in many cases. Unfortunately, programming skills are required if specialised tasks are to be performed which is not the case for many other software packages. Computer software is written which caters for most requirements, but in the case of 3D visualisation, the unpredictability of the data makes the creation of a generic software package difficult. More software is likely to be written for the purpose of making 3D development easier (in the same way that programs like Microsoft FrontPage have made Web page creation a simple task). Programmes that convert coordinate data into 3D models will become more commonplace. This will encourage development, but design of advanced features are likely to be left for the specialist programmer for some time to come.

For many applications, 3D visualisation provides a practical means for analysing data. The versatility of the tools that can be provided means that very few limitations exist. Most limitations exist only in cases where the display of the model is impractical.

If so, can it be suitably used as an Internet application?

The primary objective of the project was to create a 3D visualisation tool that could be accessed via the Internet and used for analysis. The results of this thesis show that it is possible to create such a system, however not all visualisation packages are suited to Internet use. VRML 2.0 was used in this project and was shown to be very suitable for Internet use, particularly for displaying 3D scenes. In order to enable user based interaction, both VRML and Java programming skills are required. The need for these specialised skills result in a limited use of VRML as an analysis tool.

Not long ago, VRML 1.0 was used only to describe static scenes and so analysis tools could not be programmed.

In the short time between the release of VRML 1.0 and VRML 2.0, progress has been made to the point of allowing almost all possible functionality to be achieved – with a little perseverance. The next generation of VRML is expected to allow for a few extra capabilities (such as representation of presence) but keep the same general structure. The framework already exists to make 3D visualisation an invaluable tool for Internet use and as people become more aware of it's capabilities, it will become more commonly used. The limiting factor at present is the difficulty involved in creating the tools for analysis.

7.2 Recommendations

7.2.1 Missing Data

The quality of data used in this project was very good. DEM points, contours and ortho-images were accurate and available for all the footprints. These were used to good effect but could become more useful if supplementary data was included. The following is a suggestion of data that could have been added to the VRML files to improve their value as a research tool:

A "modern" footprint

If an equivalent (roughly similar length and depth) modern footprint was used for comparison it may have aided in interpreting the significance of the Laetoli Footprints. If it were possible to put the two side by side it may be possible to notice similarities (or differences) by applying the same visualisation techniques to both.

Data for such a footprint was available (recorded using digital photogrammetry and a footprint in sand) but was not used. It can be added at a later stage if there is a request for it's inclusion.

Condition Reports

This is information pertaining to each footprint and to the site in general recorded on site by specialists involved with the project. The condition reports show areas where roots and insects caused damage to the condition of the tuff surface, and various other information of scientific value (Rüther, 1997). This information could have been overlaid on the model of the footprint in the form of line drawings so as to maintain an acceptable degree of performance. Unfortunately this information was received too late in the project development to be included but may be added in the future.

7.2.2 Missing Features

There are numerous possibilities of functionality that could have been added. More mathematical tools in particular could have been included to measure values such as the centre of gravity, volume and shape of each print. The features available at present are more than adequate for the casual observer, but the specialist may require additional features. Without a background knowledge of archaeology or anthropology it is pointless trying to predict the more complex operations that would be required. Specific tools need to be requested from someone in the know how. Perhaps this is more the weakness – that someone specialised in the scientific analysis of the footprints was not more involved in deciding what the capabilities of the visualisation package should be.

Voronoi Triangulation

The Visual Basic programmes written to convert coordinates into VRML files were designed to accept regular grids of points only. If the coordinates are not in a regular grid format, a more complicated procedure is needed to specify how the points are to be triangulated. Voronoi triangulation does this, and a particular method of Voronoi triangulation (Bowyer-Watson algorithm) was found which may have been suitable for this task (Rebay, 1993). The principle behind this algorithm (and other Voronoi algorithms) is easy to follow, however implementing it is very time consuming. This is also a possible area of future development.

If such a method was programmed, it would enable other 3D data to be converted into VRML format. An advanced type of triangulation method could even allow true 3D shapes to be converted to VRML files. Of course, the tools created using Java (measuring tools and so on) are independent and would have to programmed specific to the data, but the basic creation of VRML files for any point scatters could be produced using a generic program.

References

Abouaf J. (1999), The Florentine Pietà: Can Visualization Solve the 450-Year-Old Mystery?, IEEE Computer Graphics and Applications, Potel, M. (ed.), Jan/Feb, pp 6-11.

Anderson D. "Graphics Cards" Online, http://www.pctechguide.com/05graphics.htm. 1999.

Brodlie K.W, Carpenter L.A, Earnshaw R.A, Gallop J.R, Hubbold R.J, Mumford A.M, Osland C.D, Quarendon P. (1992), Scientific Visualization: Techniques and Applications, Springer, New York.

Burd T. "CPU Info Center" Online, Berkeley University: Department of Computer Science, http://bwrc.eecs.berkeley.edu/CIC/. 1999.

Carey R, & Bell G. (1997), The Annotated VRML 2.0 Reference Manual, A-W Developers Press, New York.

"Creative Labs: Products: Graphics: Graphics Blaster RIVA TNT: Technical specifications". Online, Creative Labs, www.soundblaster.com/hotgraphics/specs.html. 1999.

Greene K. (1983), "Archaeology an Introduction", Batsford Publishing.

Hull S. (2000), Digital Photogrammetry for Visualisation in Architecture and Archaeology.

Hartman J, & Wernecke J. (1996), The VRML 2.0 handbook – Building moving worlds on the web, A-W Developers Press, USA.

Kraus K. (1997), Photogrammetry – Advanced Methods and Applications, Vol. 2, 4th edition, Institute for Photogrammetry and Remote Sensing, Vienna University of Technology, Dümmler/Bonn.

Landes S. (1998), Impact on 3D modelling from the design of a web-based 3D information system, International Archives of Photogrammetry and Remote Sensing, Vol. XXXII, Part 5, Hakodate, pp. 339 - 343.

Lea R, Mastuda K, Miyashita K. (1996), Java for 3D and VRML Worlds, New Riders Publishing, Indiana.

Morell V. (1995), Ancestral Passions: the Leakey Family and the Quest for Human Kinds Beginnings, New York, Simon & Schuster.

Naughton P. "Java History" Online, http://ils.unc.edu/blaze/java/javahist.html. 30 Aug 1999.

Newmarch J. "Java – Myth or Magic" Online, http://pandonia.canberra.edu.au/java/auugjava-/paper.html. 1997.

Palamidese P, Betro M, Muccioli G. (1993), The Virtual Restoration of the Visir Tomb, Proceedings Visualization '93, Nielson G. & Bergeron B. (ed.), pp 420-423.

PC Webopedia, "Online Dictionary" Online. Available: http://webopedia.internet.com/, 20 Jan 2000.

Rebay S. (1993), Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm, *Journal Of Computational Physics*, Academic Press, Vol. 106, pp. 125-138

Rozendaal R. (2000), Interactive Visualisation of the Laetoli Footprints using 3D Graphics.

Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorenson W. (1991), Object-Oriented Modeling and Design, Prentice-Hall

Rüther H, (1996), Reports on the 1995 Documentation Campaign of the Laetoli Footprints, Reports Presented to the Getty Conservation Institute.

Rüther H, (1997), Reports on the 1996 Documentation Campaign of the Laetoli Footprints, Reports Presented to the Getty Conservation Institute.

Sims D. (1997), Archaeological models: Pretty Pictures or Research Tools?, IEEE Computer Graphics and Applications, Potel, M. (ed.) Jan/Feb 1997, pp 13-15.

Taek J. "Optimal Grid Generation Techniques" Online. Department of Mechanical Engineering Carnegie Mellon University, http://www.me.cmu.edu/faculty1/shimada/cg97/taek/. 1997.

Thalmann N.M, & Thalmann D. (1991) , New Trends in Animation and Visualization, John Wiley (ed.), Chichester.

Thomas G, & Finney R. (1992), Calculus and Analytical Geometry, 8th Edition ,Addison-Wesley, USA.

Woodwark J. (1991), Reconstructing History with Computer Graphics, IEEE Computer Graphics and Applications, Jan, pp 18-20.