Creating Surface Models in Virtual Reality

by Luis Casanueva

Computer Science Honours 1996

Rhodes University

Creating Surface Models in Virtual Reality

by Luis Casanueva

Computer Science Honours 1996

Rhodes University

Creating Surface Models in Virtual Reality

Submitted in partial fulfilment
of the requirements of the degree
of
Bachelor of Science (honours)
of Rhodes University

by

Luis Casanueva

November 1996

Abstract

The need for modelling objects into a three-dimension representation exists. We investigate the design of a surface modelling tool to map real objects, in real time. Emphasis is placed on achieving a simple yet effective method. Capturing the topology of the object is done via a six-degree-of-freedom input device. A Virtual Reality system is used to stress the interactivity and intuitiveness of the tool.

Table of Contents

Chapter 1

Introduction

1.1. Solid Modelling

Solid modelling involves the creation and manipulation of three-dimensional (3-D) objects on a computer. These objects are used in applications such as Computer Aided Design (CAD), Virtual Reality (VR), three-dimensional graphics, and animation.

The object information required may depend on the application. CAD may require information about the geometry of the object. A physical simulation may also need to know the volume, density or centre of mass, while a sculpting application may only require vague information about the shape of the object (as would be the case if a sculptor just wanted a rough object on which to sculpt). These various requirements can lead to a differentiation between functional solid modelling at one end, and free-form solid modelling at the other [Miller 1986]. The main idea behind functional solid modelling is the requirement of a wide range of extremely precise information. On the other hand, in free-from modelling only the aesthetics of the final shape matters [Gain 1996].

1.2. Solid versus Surface Models

[Foley *et al* 1993] describes a model in general as "a representation of some (not necessarily all) features of a concrete or abstract entity. The purpose of a model of an entity is to allow people to visualize and understand the structure or behaviour of the entity, and to provide a convenient vehicle for 'experimentation' with ... the model".

[Foley 1993] goes on to define a solid model as a closed volume bounded by planes, surfaces, or any other structure. Its boundary must be continuous and closed. No holes or improper intersections are allowed. This puts a restriction on the design of a tool intended to create solid models.

Sometimes one needs to simplify the structure of the modelled entity to make the model easier to visualize or manipulate. This leads us to the definition of a surface model. A surface model consists of a collection of separate surfaces representing parts of the object. An analogy would be that of putting 'stickers' on an object, with the 'stickers' representing the surface patches. Surface models do not need to represent the complete topological information of the object. This means that a surface model does not necessarily need to be a closed object and alleviates some of the restrictions in creating a tool to represent a model in 3-D. Once a surface modelling tool is achieved, one can create a solid model out of this surface model. There are many algorithms available that will translate a surface model into a solid model [Sheng and Meier 1995], [Thomas 1984], [Hoppe *et al* 1992], [Wyvill *et al* 1986]. This allows us to limit our research to generating a surface model. Hopefully, this will result in a product that is easier to use and to implement.

Using surface models means that we are not interested in the geometry of the object anymore, but only in its general shape. The shape should be enough to visualize the object and to use it in VR applications.

1.3. The Need for Solid Models

As stated earlier, solid models are widely used in CAD, 3-D graphics and VR applications, amongst others. The decrease in price of high-end graphics hardware has caused an increase in graphics applications, specifically VR systems [Watkins 1994]. These applications need solid models. This has caused an increase in the number of tools for creating solid models. However, these tools generally have a non-intuitive and non-interactive interface. This is due to the fact that input devices are usually two-dimensional, and this two-dimensional data must be forced into a three-dimensional representation [Sachs 1991].
This could change with recent developments in computer interface hardware. 3-D input devices and Head Mounted Displays (HMD) should give enough flexibility and versatility to allow the creation of user-friendly applications.
The creation of powerful tools can be achieved by entering information directly in 3-D using six-degree-of-freedom sensors, thus eliminating the need for 2-D-to-3-D conversion. The resulting object would then be visualized directly using a stereoscopic display.

The early 3-D input devices used mechanical linkages attached to measuring devices [Clark 1976], [Roberts 1966]. This type of device has been used until fairly recently [Foley 1987]. However, the preferred system today uses devices employing magnetic fields to accurately give the position and orientation of a

sensor in space. These eliminate any need for mechanical linkages or sonic, laser or optical sensors, making the device easier to use. One such device is the Polhemus 3Space system. This is the input device chosen for this project.

1.4. Virtual Reality

As [Gain 1996] states, there are many definitions of a Virtual Reality (VR) system. However, a term that seems to be present in all of them is **interaction** (the environment responds to the actions of the user in real time). Since one of the aims of VR is to simulate reality, users usually interact with the VR system as normally as they would in the real world. This provides the advantage that the user already has the knowledge to operate in a real-life environment. If the environment was markedly different from real life, the user would have to learn how to interact with it. Therefore, using a VR system in which to build our modelling tool would be ideal. It would provide access to 3-D input and output devices already used in VR systems, and would at the same time supply the user with an intuitive interface. Using a Head Mounted Display (HMD), the user can visualize the virtual object in a stereoscopic way, thus eliminating any ambiguities created by 2-D output devices. The user would also use a hand-held 3-D input device to enter object data. This adds to the intuitiveness and ease of use of the system since it relies heavily on the innate hand-eye co-ordination of the user.

The VR system available to us is the RhoVeR (Rhodes Virtual Reality) system. It provides us with various input devices. One of these devices is the Polhemus 3Space InsideTrak system. Our work is closely related to the 3-D data generated by this device.

1.5. Aim

At the moment, if RhoVeR users want a specific object, they would have to:

- Get the object from an outside source. (Sources of objects are available over the Internet.) The problem is that these sources are limited, and one might have compatibility problems with the various formats of 3-D objects available.

- Create them by inputting the co-ordinates of their vertices, edges and faces. Usually one would have to draw the object on graph paper from different points of view. This could be a very tedious process.

- Purchase a commercial system to create or digitise objects. This is usually very expensive and a small need for objects does not justify such expense.

The need to create object models in RhoVeR does exits. A tool to digitise real objects would be a nice feature to add to the package. This would also provide a testbed virtual environment. Because RhoVeR is a relatively new system, such applications can be used to test and evaluate various aspects of the system.

With this in mind, we decided to design a modelling tool, working under RhoVeR, which would allow the creation of simple surface models. Emphasis was placed on simplifying the approach, with the hope of making the system intuitive. This simplification should also decrease the latency, which is a central consideration in VR systems, as well as reducing the difficulty of the implementation of the program.

1.6. Organisation of the Thesis

The remainder of the thesis is organised as follows:

- **Chapter 2: Solid Representations.** This chapter presents the various ways of representing solids, and explains the choices made.

- **Chapter 3: The Input Device and its Data.** This chapter presents the input device used and the information derived from it**.**

- **Chapter 4: Creating Surface Model Tools.** This chapter presents the creation of two modelling tools. It also covers minor issues concerning these tools.

- **Chapter 5: RhoVeR: The Rhodes Virtual Reality System.** This chapter presents the VR system used, and its integration with the modelling tools.

- **Chapter 6: Conclusion.** This chapter presents the concluding remarks and suggests future work.

- **Appendix A: Examples of Surface Models.** This appendix shows some images of objects mapped with the two tools derived in Chapter 4.

Chapter 2

Solid Representations

After deciding on the need for a tool to map real objects into surface models, we decided to look at the various available ways of representing the model.
A solid representation is "a means of encoding the shape of a solid object" [Gain 1993]. Solid representations can be divided into various categories. Extensive research has been done in this field in the Department of Computer Science at Rhodes University (including [Gain 1993] and [Watkins 1994]). This chapter is a brief overview of that work, with references to earlier solid modelling tools.

2.1. Boundary Representations

In Boundary Representations (B-reps), the object is modelled using only its topological surface information. This usually means using polygons (vertices, edges and faces) or curved surfaces. However, it is quite difficult to determine faces if curved surfaces are used [Foley *et al* 1993]. These curved surfaces are

then approximated by polygons or surface patches. Therefore B-reps are further subdivided into Polyhedral and Non-Polyhedral.

2.1.1. Polyhedral B-reps

In Polyhedral B-reps, the object is represented using a mesh of polygonal faces. The mesh covers the entire surface of the object. The polygons in the mesh can only be planar polygons. Various levels of restrictions can be imposed on the polygon to make it a *polyhedron* or to impose a 2-manifold restriction. These issues are fully dealt in [Foley *et al* 1993] so no further discussion is required.

The advantages of Polyhedral B-reps are:

- The storage needed is minimal as opposed to the other categories of solid representations. An example of a storage method is the Object File Format (OFF).
- Geometrical transformation algorithms on polygon meshes are efficient and widely available.
- Most of the available methods for displaying objects in a computer use the polygon mesh, and thus no expensive transformation is required.

The main disadvantage of Polyhedral B-reps is that curved surfaces are only approximated by straight-line polygons.

2.1.2. Non-Polyhedral B-reps

In Non-Polyhedral B-reps, the object's surface is represented by parametric bicubic patches. These patches are generalisations of parametric cubic curves (such as Bezier and B-Spline curves) [Gain 1993]. An example would be the Non-Uniform Rational Basis-Spline (NURBS) [Gain 1996].

The advantage of this method is the accuracy with which parametric bicubic patches approximate curved surfaces.
The disadvantages are the complex and expensive algorithms needed, even for simple transformations, and the expensive need to map into a polygon-mesh for display. This is due to the fact that Non-Polyhedral B-reps create "unevaluated" models. "Unevaluated models contains information that must be further processed in order to perform basic operations" [Foley *et al* 1993].

2.2. Spacial-Partitioning Representation

2.2.1. Cell Decomposition

In Cell Decomposition, primitive solids are "glued" together along vertices, edges or faces to form solid models. The primitives can have any shape, size, position or orientation as long as they do not intersect with each other.

2.2.2. Spatial-Occupancy Enumeration

This method is a variation of Cell Decomposition, with the solid primitive being restricted to a specific shape (usually a cube) orientation and size. Only the position of the primitive is left unrestricted. The primitive is called a VOXEL (Volume Element) as an analogy to PIXEL (picture Element) [Foley et al 1993].

The advantages of Cell Decomposition and Spacial Occupancy Enumeration are the ease and swiftness in determining if a point lies inside or outside a solid, and the fact that any solid can be represented using these techniques.

The disadvantages are the great storage space required and the fact that it needs to approximate the surface of the object. The VOXELS cause very rough approximations to curves or even straight lines. This problem is similar to the aliasing problem encountered in 2-D displays. This can only be improved by an increase in resolution (i.e. a decrease in the size of the primitive solid). This causes a huge increase in the storage space required for the object: halving the size of the primitive would increase the number of VOXELS by eight if the primitive is a cube.

2.3. Constructive Solid Geometry (CSG)

CSG is also derived from Cell Decomposition. Solid primitives are now allowed to intersect with each other. We allow Regularised Boolean Set Operations (RBSO) (union, intersection, difference) and affine transformations (translation, rotation, scaling) to be applied to the primitives [Gain 1993]. The model is usually kept in a

tree-like structure with the primitives at the leaf nodes and the RBSOS or affine transformations in the internal nodes.

The advantages of CSG are:

- CGS can be represented in a very compact manner by using a tree structure.
- Using recursive algorithms, one can evaluate the tree structure in a simple manner.

The disadvantages of CSG are:

- All operations need to evaluate the tree structure. These operations are expensive since they usually involve the use of recursion.
- "CSG creates unevaluated models "[Foley *et al* 1993]. The object must usually be transformed into a polygon mesh structure to be displayed. This process is computationally expensive.

2.4 Selected representation

We decided to select a polygon-mesh representation for the following reasons:

- As opposed to most of the other methods that require extensive and expensive pre-processing, the polygon-mesh representation needs little pre-processing before display. This is significant since speed is an important requirement in VR systems.
- Rich and efficient algorithms for visualization and rendering are widely available.

- A polygon-mesh representation is simple to visualize and has plenty of efficient transformation algorithms. This is crucial in our aim to simplify the design of the surface modelling tool.
- There are various cheap and concise formats available for storing a polygon-mesh model. One such example is the Object Format File (OFF), as used by the RhoVeR system.

The only drawback of polygon-mesh models is the loss in precision when approximating curved surfaces. This can be a problem in functional solid modelling applications such as CAD. However, we are not interested in mathematical precision. A "good enough" approximation of the curved surfaces will suffice for our tool.

2.5 Past research in modelling

This section is a quick overview of various interesting research papers on creating solid models from physically captured data.

[Hoppe *et al* 1992] use a mixture of polyhedral B-reps and Spatial Occupancy Enumeration to construct solid models. In their approach, they only use an unorganised collection of points. Although their approach has an incredible amount of merit, we found that information given by 3-D devices like the Polhemus InsideTrak would have alleviated much of the computation of the algorithm.

[Sheng and Meier 1995] used non-polyhedral B-reps to generate topological structures for Surface Models. Given a surface model, they are able to generate a topological structure for that model which includes a closed volume with a continuous, closed boundary. Their work is very relevant to this work because it allows us to restrict our approach to just creating a surface model instead of a solid model.

[Loop 1994] used Smooth Spline Surfaces (i.e. non-polyhedral B-reps) to re-tile irregular meshes with aesthetically pleasing results. If a mesh is regular, the resulting surface is equivalent to a B-spline. Otherwise, the resulting surface has a degree 3 or 4 parametric polynomial representation.

[Galyean and Hugues 1991] used Spacial Occupancy Enumeration to create an interactive volumetric modelling tool. They used a Voxmap (a data array), in analogy with a Bitmap, to store the object. As an input device, they used the Polhemus Isotrak device. They also constructed their own output device to create force feedback, representing the approach of the tool to the surface.

2.6. Conclusion

As seen in this chapter, there are various ways to represent a solid in computer graphics. All have advantages and disadvantages. Some of them are more complicated than others, but offer more powerful solutions to specific problems. After an overview of the different methods and the research in those specific methods, we felt confident enough to choose the B-rep polyhedral representation. This was mainly due to the few disadvantages in polygon-mesh modelling. B-rep

polyhedral representation also has the advantage of being closely associated with points and vectors. These are familiar linear algebra concepts, so there was no need to learn new concepts.

Incredibly enough, we found in the literature that most of the research is done using other representations. This might be because a lot of research has already been done in polygon-mesh modelling, and no really innovative modelling tool can be found using this technique. As we will try to demonstrate, 3-D input devices and a simple approach might prove the last statement wrong.

Chapter 3

The Input Device and its Data

After choosing the use of the B-reps polyhedral representation method, we decided to fully investigate the input device available to us, as well as the meaning of the data that is given to us by that device.

3.1. Input Devices: The Polhemus 3Space system

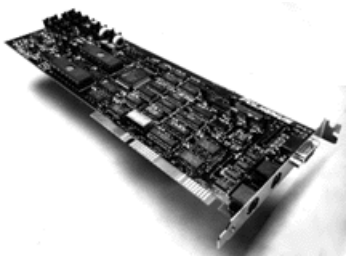

Figure 3.1: The Polhemus InsideTrak ISA board.

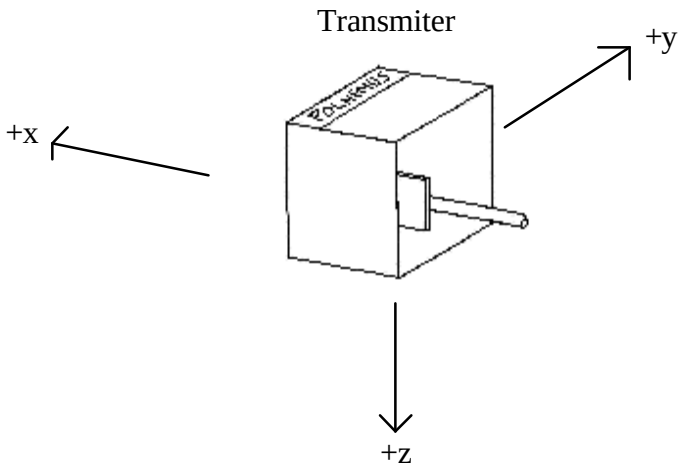

Figure 3.2: Transmitter for the Polhemus InsideTrak system.

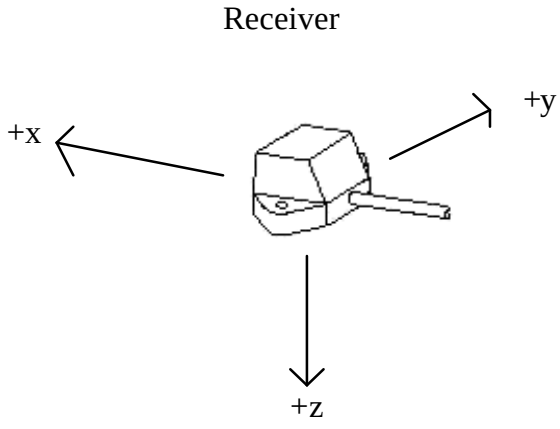

Figure 3.3: Receiver for the Polhemus InsideTrak system

The Polhemus 3Space InsideTrak system is available to us. It is composed of an ISA card that plugs into an IBM compatible personal computer (PC) (figure 3.1), a transmitter (figure 3.2), and one or two receivers (figure 3.3). The system uses magnetic fields to accurately compute the position and orientation of the receiver, with respect to the transmitter, as it moves through 3-D space. This provides us with dynamic six-degree-of-freedom measurements of position (X, Y and Z Cartesian co-ordinates) and orientation (azimuth, elevation, and roll). These measurements are provided in real time, and the data can be updated continuously or discretely (point by point). The data has high resolution and accuracy, and low latency. The Polhemus InsideTrak does however have some drawbacks. It uses magnetic fields and thus gives erroneous readings if a metallic object is nearby. This means that we will not be able to digitise metallic objects. Also, the receivers and transmitter are connected to the ISA board with wires. This can cause the manipulation of the Polhemus InsideTrak to be awkward. This last problem has been solved by Polhemus with the recent release of a wireless 3Space tracker: the STAR*TRAK system. A limited range of 76.2 cm [Polhemus 3Space 1993] between the transmitter and the receiver can also cause problems if the user wants to digitise large objects. This has also recently been solved by Polhemus with the release of the LONG RANGER device that, when added to the InsideTrak system, multiplies the normal range by a factor of three [Polhemus 3Space 1996].

In this thesis, we will refer to the Polhemus 3Space InsideTrak system as the *Polhemus tracker*. We will refer to the receiver of the Polhemus InsideTrak system as the *tracker*.

3.2. The Information given by the Polhemus tracker

The Polhemus tracker gives us the following information:

- Position of a point in 3-D space (X, Y, and Z).
- Orientation of the tracker in Euler angles (Azimuth, Elevation, and Roll) (see figure 3.4, 3.5, and 3.6) or as a quaternion..

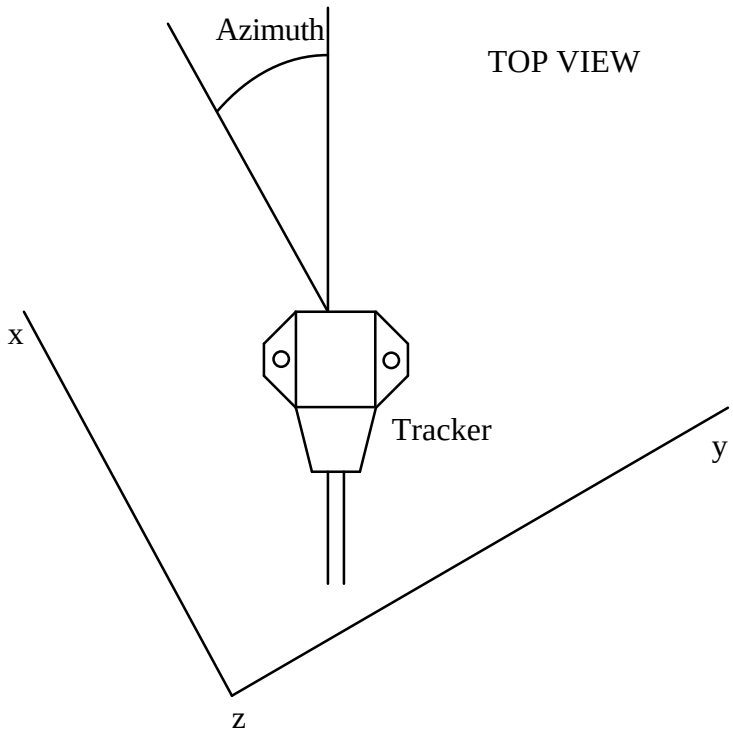

Figure 3.4: The Azimuth information given by the Polhemus tracker.

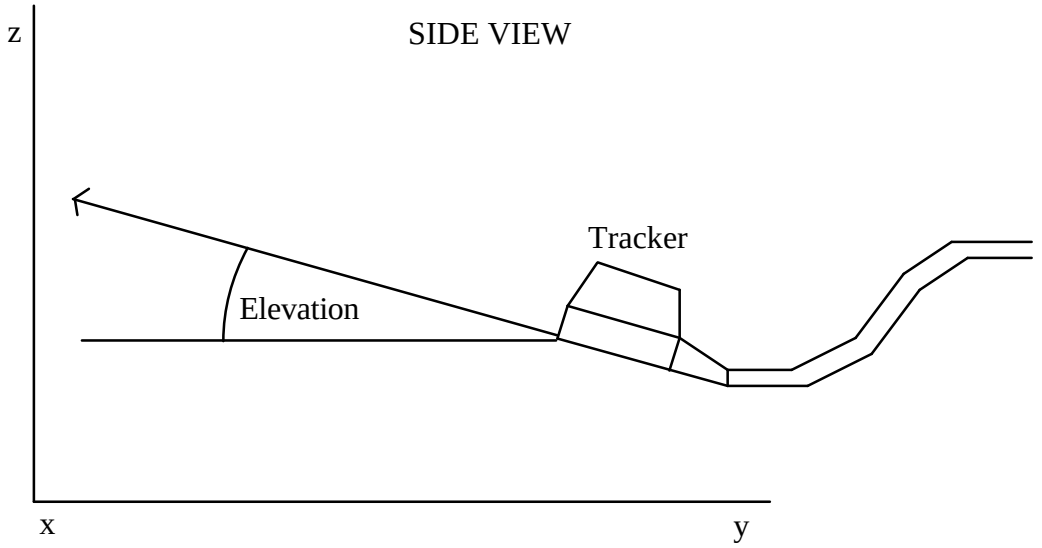

Figure 3.5: The Elevation information given by the Polhemus tracker

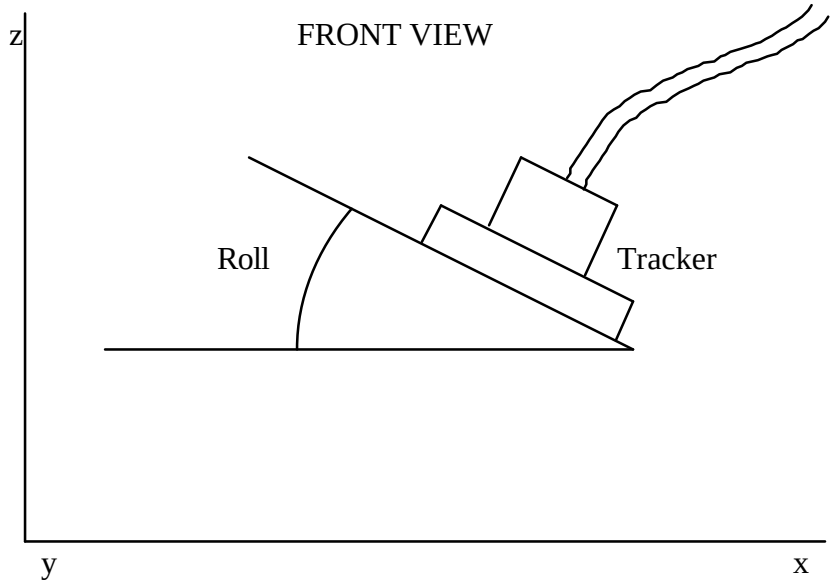

Figure 3.6: The Roll information given by the Polhemus tracker.

This data allows us to extract the following information:

- The position of a point in 3-D space (X, Y, and Z).
- The normal vector to a plane containing the point, and corresponding to the base of the tracker.
- The "**forward**" vector, belonging to the plane, and corresponding to the direction the tracker is pointing in.
- The sequence in which the data is captured.
- The "**direction**" vector in which the tracker is moving (note that this **direction** vector is different from the **forward** vector).

3.3. Extraction of the data

As stated earlier, one can obtain a great deal of information from the data that the Polhemus tracker provides. In this thesis, we will refer to the set of data given by a specific pulse as the **current** set of data. The set of data before the **current** set of data will be referred to as **previous**, while the set of data after the **current** set of data will be referred to as **next**.

3.3.1. The point

The Polhemus tracker gives information about a position in 3-D space, thus providing a 3-D point. No manipulations are required here.

3.3.2. The **forward** vector

The Polhemus tracker gives information about the direction as Euler angles (Azimuth, Elevation, and Roll). From the Azimuth and Elevation data, one can get the **forward** vector as follows:

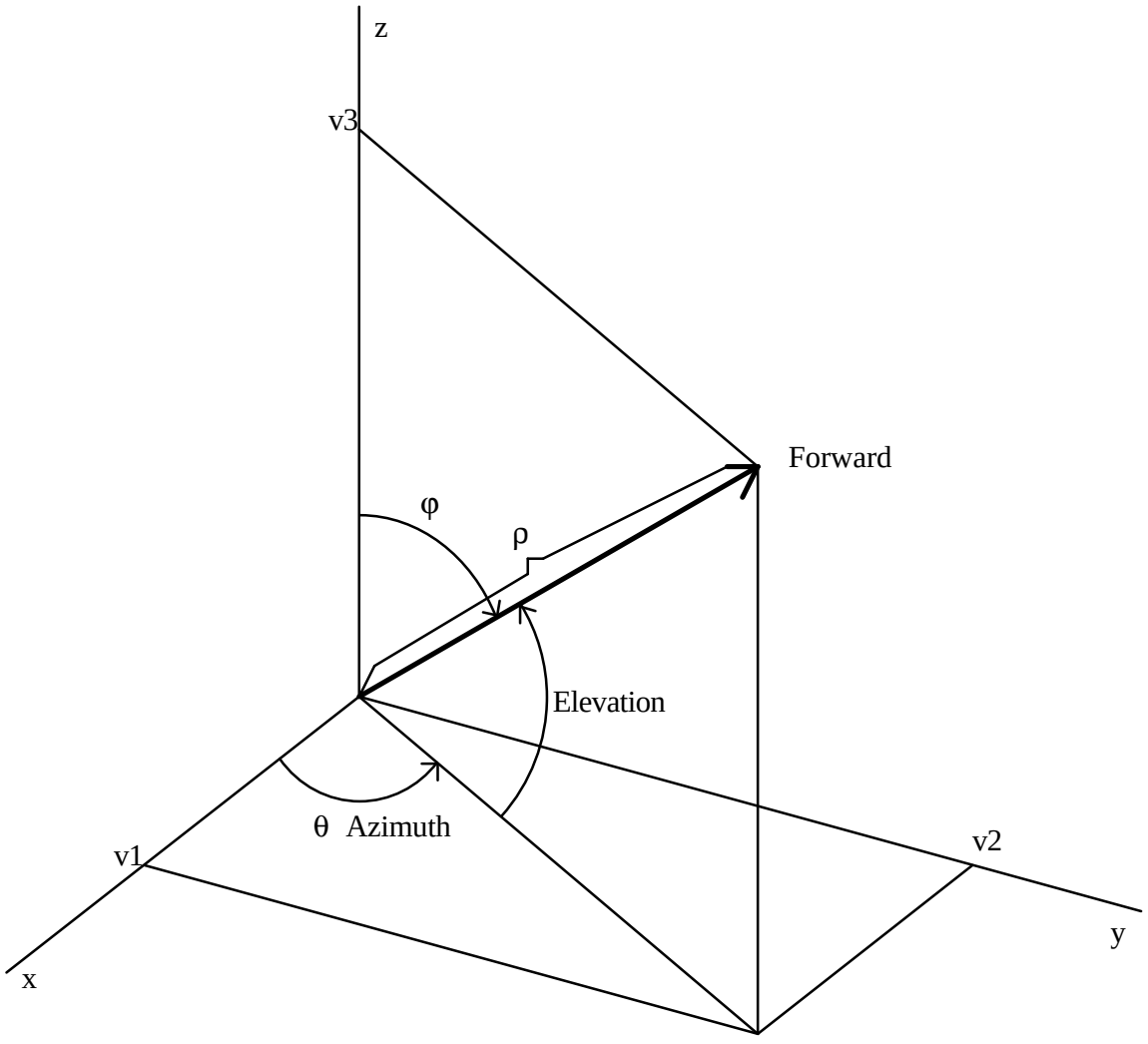

Figure 3.7: The **forward** vector in Euler angles.

As you can see, figure 3.7 provides us with the co-ordinates for v1, v2, and v3 [Ammeraal 1986].

$v1 = \rho \sin \varphi \cos \theta$

$v2 = \rho \sin \varphi \sin \theta$

$v3 = \rho \cos \varphi$

where $\varphi = (90° - \text{Elevation})$ and $\theta = \text{Azimuth}$.

But $\sin (90° - \text{Elevation}) = \cos \text{Elevation}$ and

$\cos (90° - \text{Elevation}) = \sin \text{Elevation}$

Since we are not interested in the magnitude (the length) of the Forward vector (ρ), we can set it to 1. This gives us the co-ordinates of the **forward** vector:

v1 = cos Elevation cos Azimuth

v2 = cos Elevation sin Azimuth

v3 = sin Elevation

3.3.3. The normal vector

Similarly, one can find the normal vector. First let us visualize what Roll is (see figure 3.8).

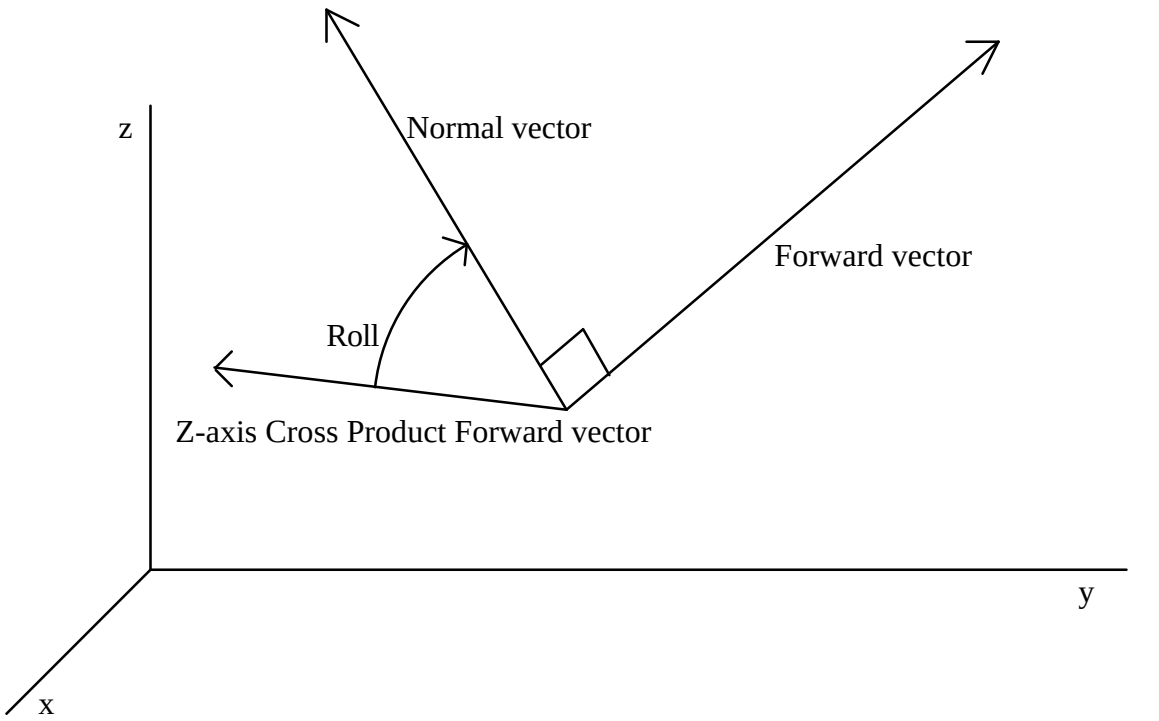

Figure 3.8: The Roll angle given by the Polhemus tracker.

One must first get the vector resulting from the cross product of Z-axis (i.e. [0, 0, 1] and **forward**. The Roll angle is the angle formed by the vectors **normal** and (Z-axis CrossProduct **forward**), in the plane formed by these two vectors. One can then get the co-ordinates of the **normal** vector.
This is, however, a fairly complicated procedure.
Since getting the normal vector from Euler angles is awkward and computationally expensive, we decided to look for another cheaper and simpler method.

The Polhemus tracker can also be set to give the information as a point and a quaternion. Using the quaternion, one can get the rotation matrix. As this operation

was already available in the RhoVeR software package, we decided to use this instead of using Euler angles.

Since we have no constraints about whether the horizontal plane in real life must correspond to the horizontal plane in the virtual world, we can assume that at the start the tracker is resting horizontally. The normal vector is therefore [0, 0, 1]. We can then apply the rotation matrix to this vector to get the normal vector of the tracker.

If one wanted the real horizontal plane to correspond to the virtual horizontal plane, one would just need to apply a rotation to the data.

One can also get the **forward** vector using the same method.

3.3.4. The sequence of data

The information about the sequence in which the data is entered does not need any manipulation. This information is reflected in the creation of a square to represent a plane. As will be shown later, we used information about the previous set of data as well as the next set of data to create a square. The sequence of data was also used to get the **direction** vector.

3.3.5. The **direction** vector

One can also get the **direction** vector. This vector is the vector tangent to the path of the tracker at a specific point. It is important not to confuse this vector with the **forward** vector. The **forward** vector is related to the orientation of the tracker, while the **direction** vector is dependent on the path of the tracker (see figure 3.9).

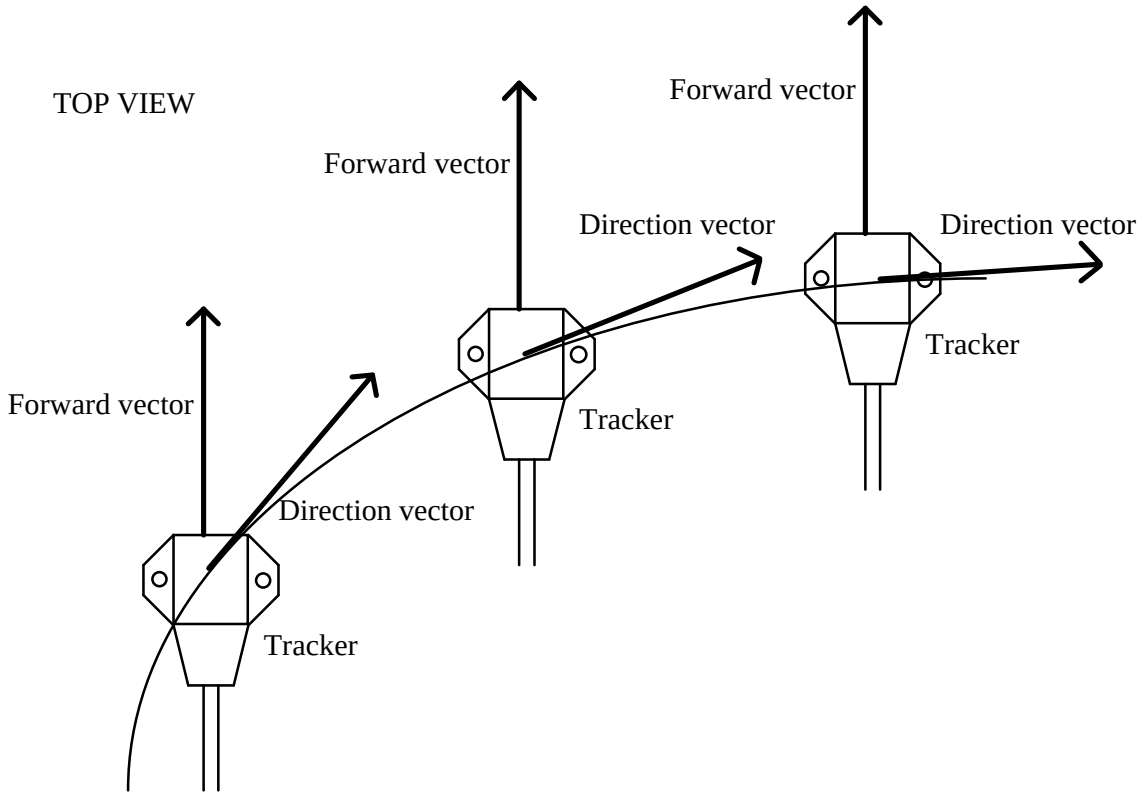

Figure 3.9: Difference between the **forward** and the **direction** vectors.

Calculating the **direction** vector, however, can prove very complex, as one needs to create a mathematical equation that will recreate the path of the tracker. Thus, we settled for an approximation that would give us enough information to be useful, without increasing the complexity of the algorithm. Three ways come to mind immediately:

The vector between the **previous** point and the **current** point (see figure 3.10).
The vector between the **current** point and the **next** point (see figure 3.11).
The vector between the **previous** point and the **next** point (see figure 3.12).

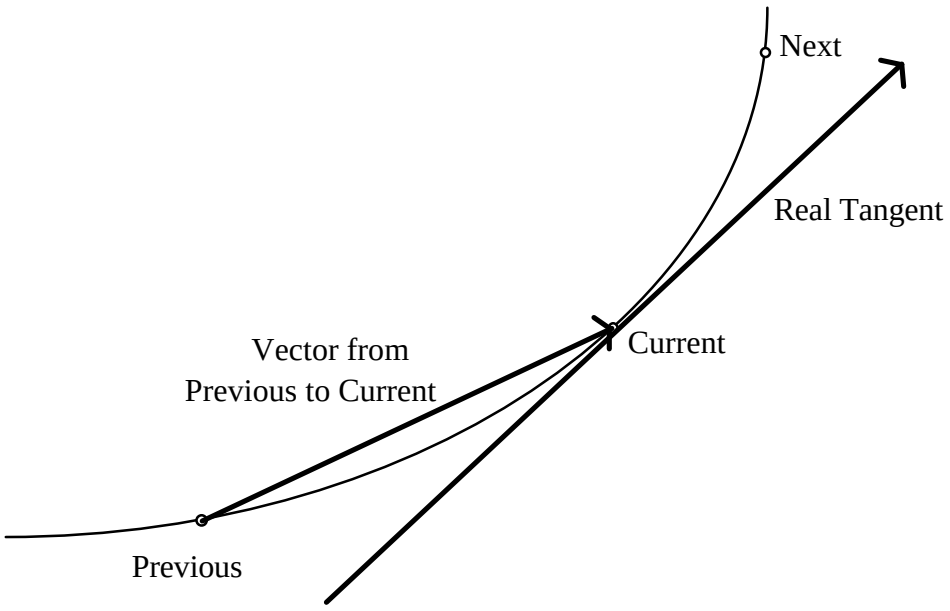

Figure 3.10: Tangent to the curve at point **current** and vector between points **previous** and **current**.

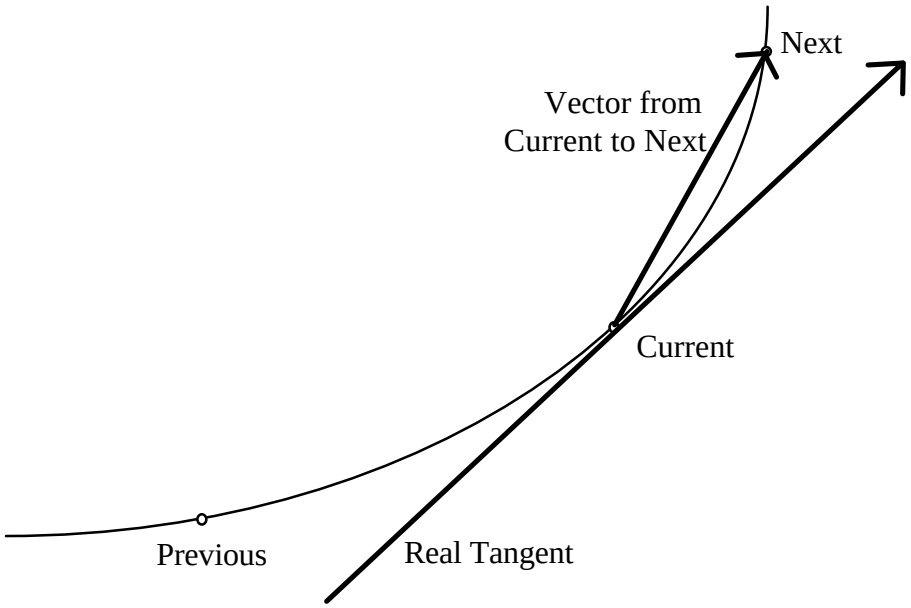

Figure 3.11: Tangent to the curve at point **current** and vector between points **current** and **next**.

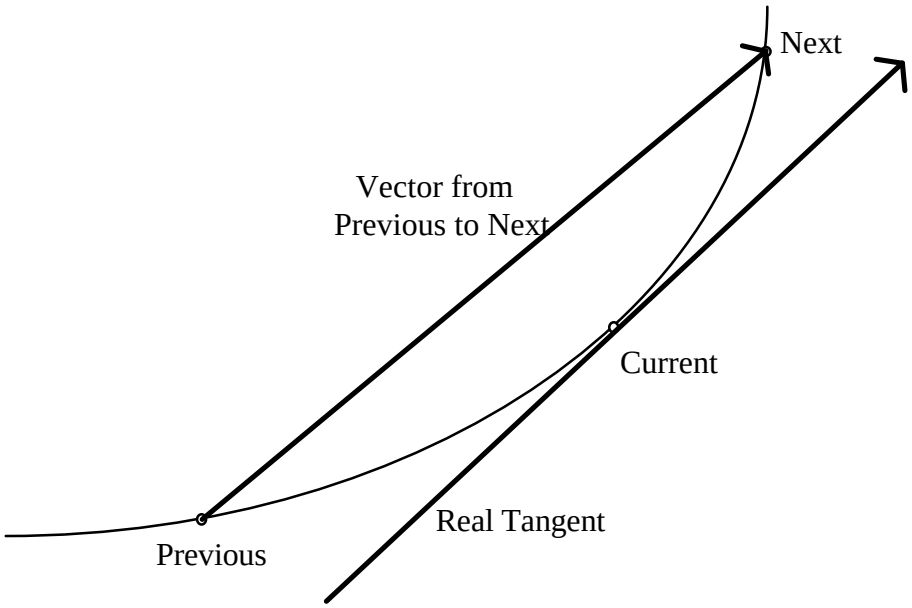

Figure 3.12: Tangent to the curve at point **current** and vector between points **previous** and **next**.

From figures 3.10 to 3.12, one can clearly see that the vector between the **previous** and the **next** point is the best of the three. One can also see that it actually is quite a satisfactory approximation to the tangent.

There are however cases where the approximation is not so good. These cases arise from the inconsistent use of the tracker by the user. If the user uses abrupt accelerations, the resulting approximation will lose accuracy (see figure 3.13). This is made worse if the user also uses sharp turns (see figure 3.14).

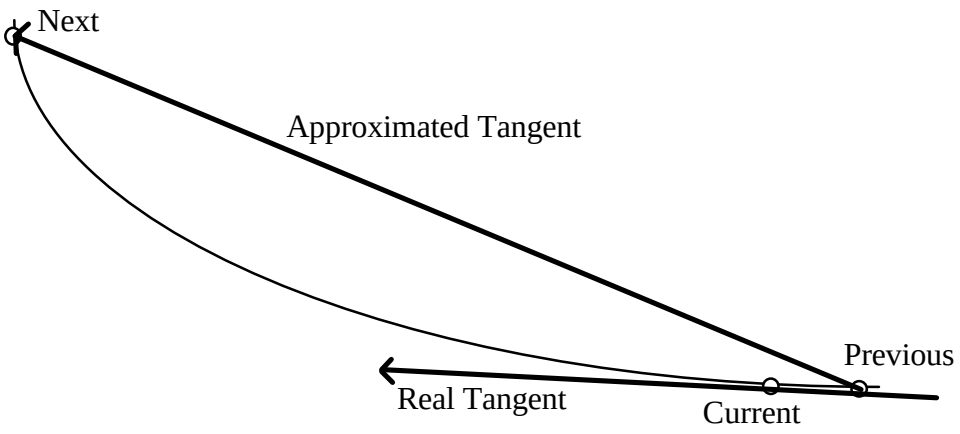

Figure 3.13: The resulting approximation to the tangent with a large acceleration of the tracker.

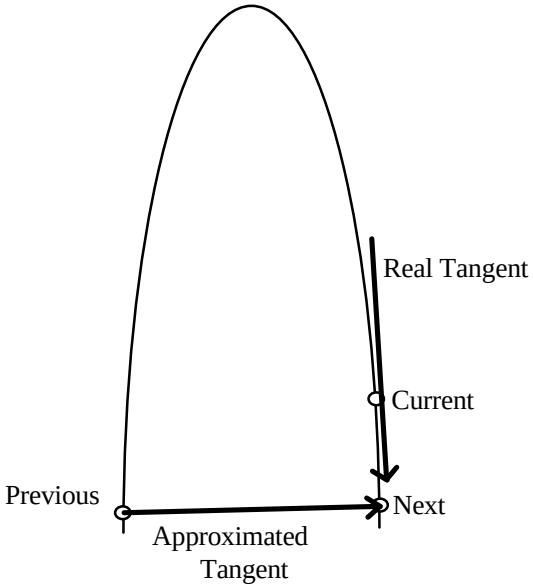

Figure 3.14: The resulting approximation to the tangent with a large deceleration and a sharp turn of the tracker.

Assuming that the movement of the tracker is kept relatively constant, the approximation of the tangent should suit our needs quite effectively. This is partially guaranteed by the high frequency at which the Polhemus tracker sends the data (60 pulses per second with one tracker). This means that the acceleration is kept roughly constant. On top of this, one can implement a method so that points will only be accepted if they are a certain distance away from the previous point.

3.4. Options for constructing objects

The information gathered in the previous section allows for various options in constructing a surface model.

3.4.1. The surrounding sphere

Imagine having a virtual sphere around a real object, loosely surrounding it. One can find a mapping between a point in the sphere and the point given by the tracker (see figure 3.15). In this way, one could map the sphere onto the object by tracing the surface of the object [Gain 1996]. This method would give a closed solid model. However, it would not be able to produce solids with holes and loops like a torus (or a doughnut).

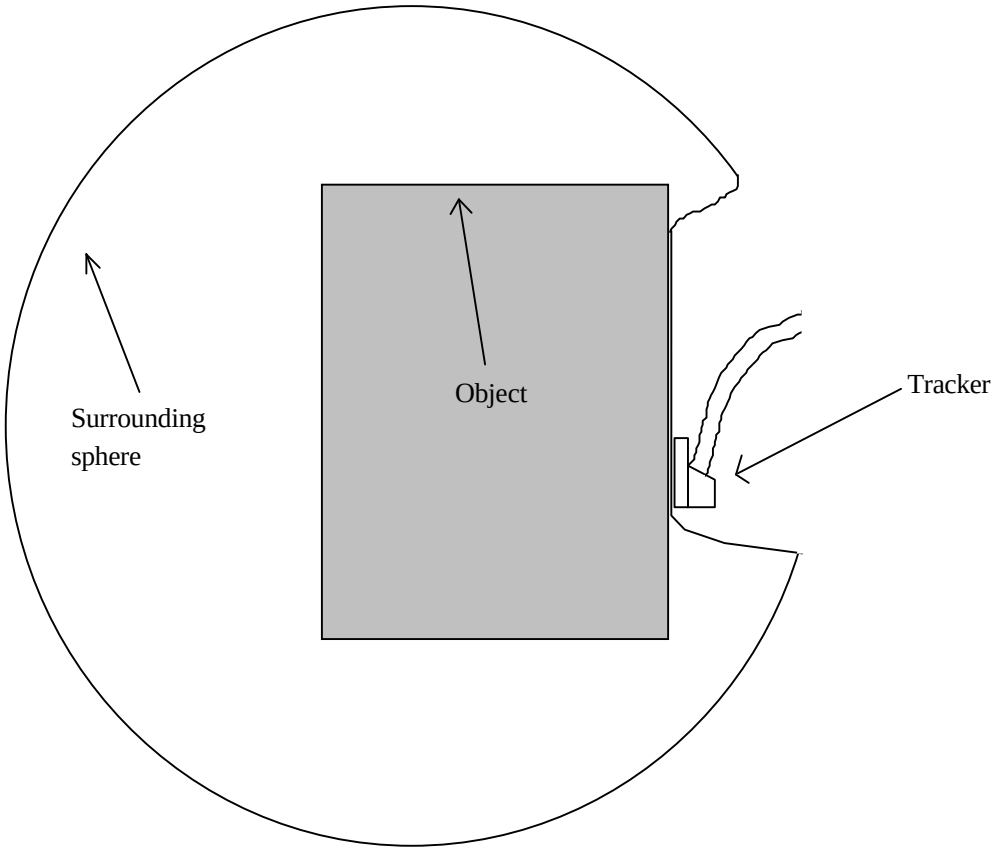

Figure 3.15: Creating a solid model by tracing an object inside a virtual surrounding sphere

A similar way to produce a solid model would be to use glove-based moulding by using a virtual clay-sculpting metaphor. One can then deform a virtual sphere moulding it to the shape of the real object [Gain 1996] (see figure 3.16).

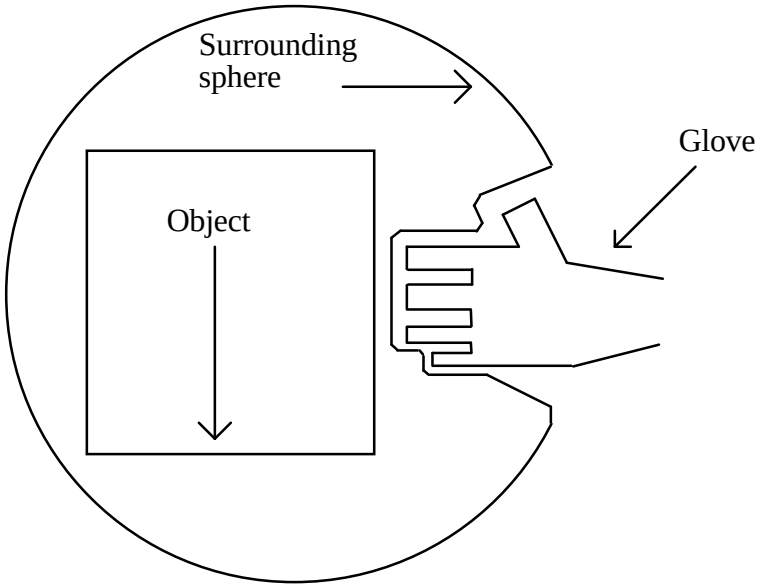

Figure 3.16: Creating a solid model by squashing a virtual surrounding sphere around a real object.

3.4.2. Cluster of points

In this case, the position of the tracker in 3-D space over time is collected. This creates a cluster, or cloud of points. If the cloud is dense enough, solid models could be created out of this cluster of points [Hoppe *et al* 1992], [Bajaj *et al* 1995]. Of all the information available from the Polhemus tracker, this method only uses the point data. If one considers the point data and the normal data to the surface at those points, the process in [Hoppe *et al* 1992] would be further simplified and hence improved.

3.4.3. 3-D curves

The Polhemus tracker can give continuous or discrete information, depending on the mode of operation selected. Both these modes would allow us to create 3-D curves. These curves could be used to build objects [Sacks *et al* 1991] or even solid models [Sheng and Meier 1995].

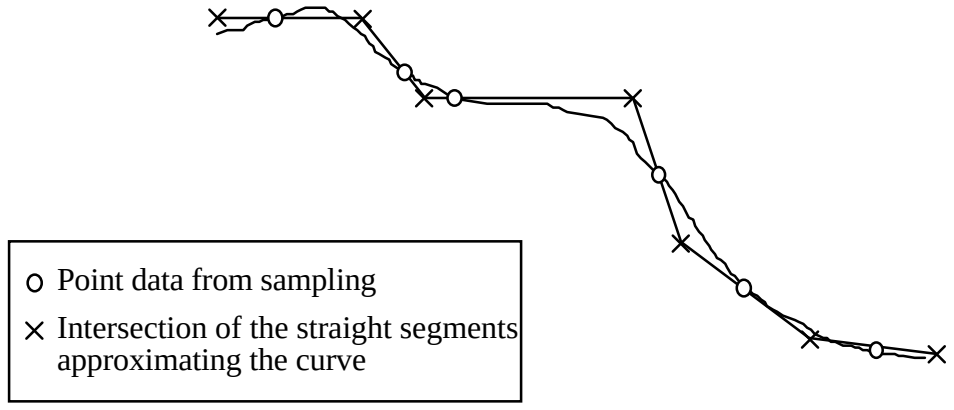

Figure 3.17: Fitting a continuous curve into discrete segments

Again, only information about the 3-D point and the order of capture of the points is used.

3.4.4. Contours

This method consists of tracing the object with the tracker so as to create parallel contours, or cross-sections of the object. The best analogy would be that of a contour map, where each contour traces the edge of a surface at a given height. These contours are made up of 3-D curves. Using triangulation techniques, one can find the optimal surface between contours [McGregor 1986]. In a simplified way, this would involve creating triangles between points of adjacent surfaces (as shown in Figure 3.18) [Miller *et al* 1991].

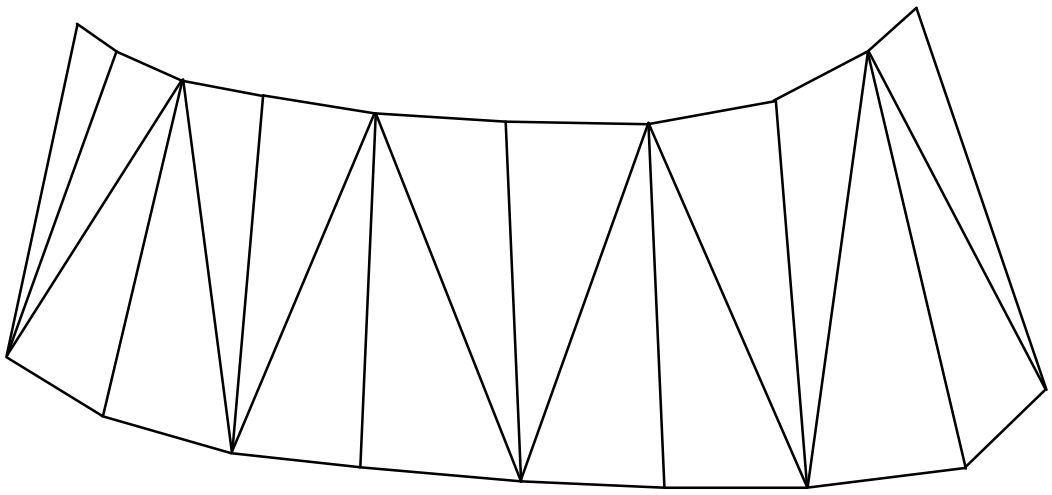

Figure 3.18: Triangulation of two curves.

However this method requires fairly parallel contours. Other similar methods are available that create surfaces from non-parallel sampling [Payne and Toga 1994]. These methods eliminate the restrictions and constraints that make the previous

algorithm awkward to use. These types of algorithms are perfect for creating solid objects from data from medical scans.

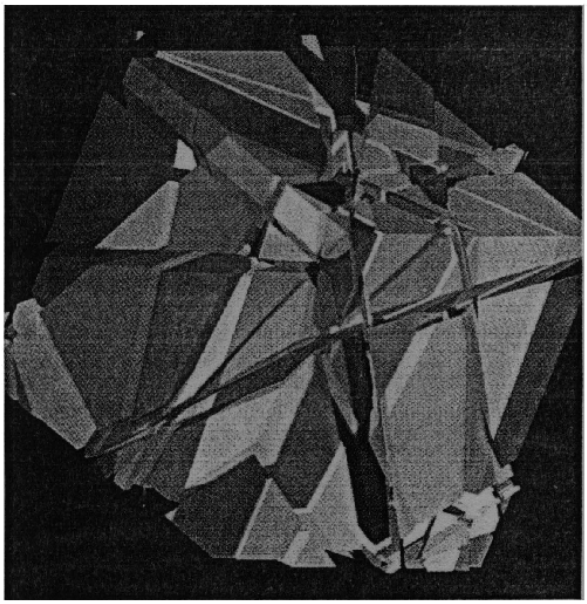

Figure 3.19: Cube formed from non-parallel contour curves [Payne and Toga 1994].

As with the 3-D curves, only the information concerning the 3-D point and the order in which the data is collected are needed.

3.4.5. Surfaces

All the previous methods made limited use of the type of data that is available to us when we use the Polhemus tracker. Creating a surface for each *pulse* of data would use most of the information given by the Polhemus tracker.

This improves the possibility of finding a more efficient way of creating surface objects. This efficiency becomes important when implementing the system with Virtual Reality where latency is one of the most important factors. Due to all these factors, we decided to choose the surface method in our investigation.

To create a surface, we can use the normal. This will give us a plane to which the surface belongs. The point is then used to position the plane in 3-D space. To represent a surface, we have decided to use squares. Although triangles are a better representation (three different points make a plane), squares give the human being a more intuitive view of the plane.

The co-ordinates of the square can be calculated using the **forward** vector and the normal as shown in figure 3.20.

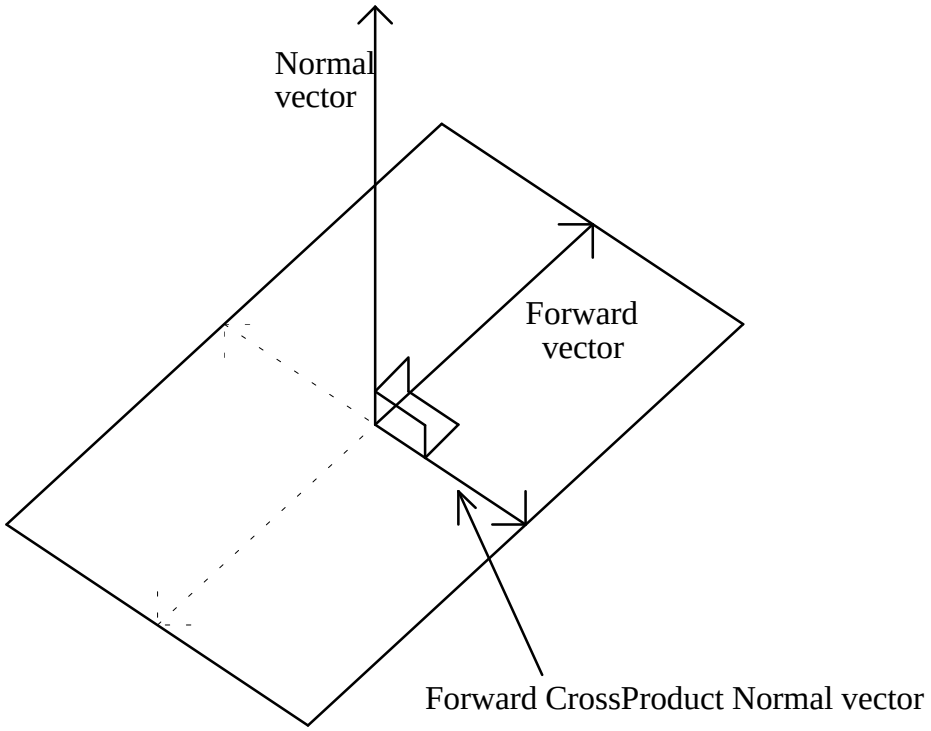

Figure 3.20: Creating the square.

The sequence in which the data is captured can be used to determine information such as the length of the **forward** vector, or which square is right in front, or behind the **current** square.

3.5. Conclusion

In this chapter, we have analysed the data given to us by the available input device: the Polhemus tracker. We have also derived ways in which to collect that data and transform it into useful information. For each *pulse* of data, we can get a

point corresponding to the position of the tracker in 3-D space, a **normal** vector corresponding to the orientation of the base of the tracker in 3-D space, a **forward** vector corresponding to the direction in which the tracker is pointing, and a **direction** vector corresponding to the tangent to the path of the tracker at the given **point**. We can also get information about the sequence in which the various *pulses* were captured.

We then looked at an appropriate way to use this information to create a surface model. We concluded that creating a square for each *pulse* would seem to be the most convenient way of representing the model.

As with Polyhedral B-reps, constructing a square representing a surface is also closely associated with points and vectors. For this reason, these two methods combine well with each other.

Chapter 4

Creating Surface Model Tools

At the end of the last chapter, it was decided that the most promising way to create surface models was through the creation of square surfaces, using most of the data from the Polhemus tracker. We also concluded on the use of squares to represent those surfaces. In this chapter, we will see how these squares can be developed into two metaphors: the "**scales**" and the "**toilet paper**". We will investigate the advantages and disadvantages of the various methods.

4.1. The "**scales**" metaphor

Creating squares to visualize a surface has given us a surface modelling tool. The object is represented by a fairly large number of small squares "glued" to the surface of the object. This looks similar to the object being covered with **scales**, hence the name for this method.

4.1.1. Creating the **scale**

A square was chosen to represent the plane in Chapter 3. Geometrically, a triangle will represent a plane better than a quadrilateral polygon because a triangle will always lie in a plane, while a higher order polygon may not. Having three points also allows us to displace any points with the result always being a planar polygon. Visually, however, a square seems to represent a plane more effectively than a triangle. This might have to do with the way humans perceive 3-D objects in 2-D devices. Humans seem to associate planes with structure based on parallel lines. This condition might be induced as a result of the person-made planar surfaces around us that are constructed out of parallel structures: Tables, building faces, doors... In school, when representing a plane in 3-D geometry, teachers tend to use a parallelogram in the blackboard (a 2-D surface). Although this representation is highly ambiguous (it could actually represent a parallelogram in 3-D), everybody seems to "see" a square. Justifying this issue is a whole problem in itself and is beyond the scope of thesis. This effect can be appreciated in figures A1 and A2 of Appendix A. The results in figure A2 could be affected by the shading, so one must ignore this.

The square that represents the **scale** is created as follows (refer to figure 4.1):

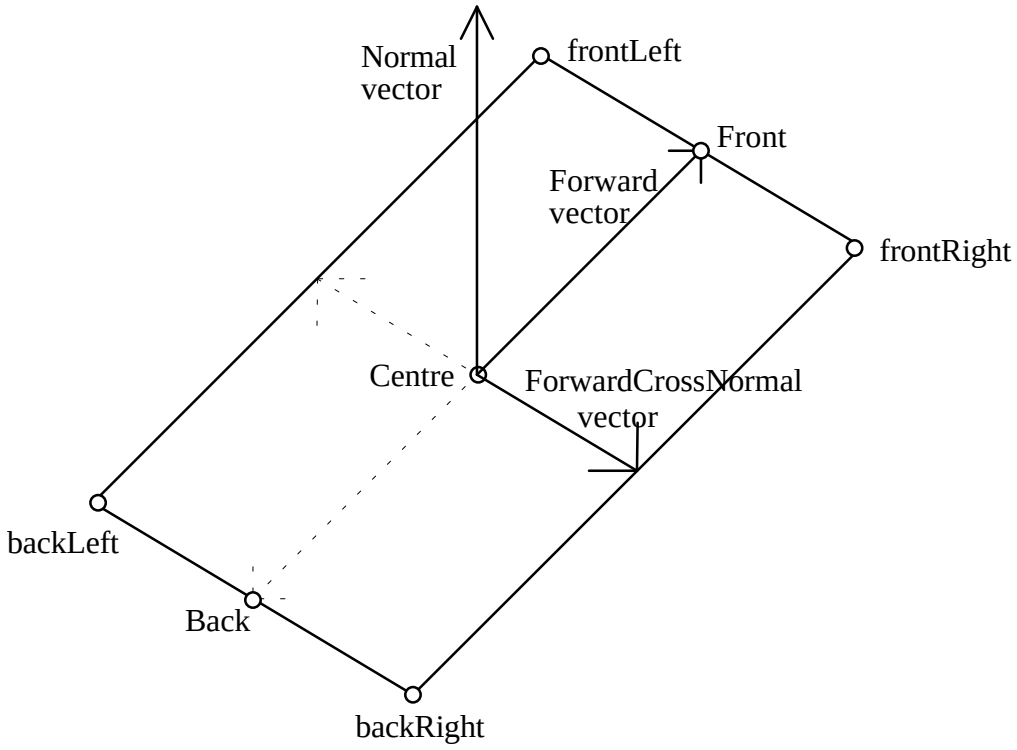

Figure 4.1: Creating the **scale** square.

We take the point, called the **centre**, and use the **forward** vector to get another point: the **front** point. The length of the **forward** vector is determined so as to approximate the distance that would be required if this square is to "touch" the next square. The correct distance is given by the "better length" in figure 4.2. The "better length" can be calculated using Pythagoras' theorem from the angle α and from half the distance between the **current** and **next** centres. The angle α is the angle between the **forward** vector and the vector from the **current** centre to the **next** centre. It can be calculated using the dot product between these two vectors. This has, however, not been implemented because such a high level of precision is not required, and it would only cause overhead in the computation. A cheaper way

to get the forward length is to use half the distance between the **current centre** and the **next centre**. This length works well when the squares are in the same plane, but the moment the angle α becomes large, small gaps start appearing between the squares. However, the approximation given by this length is good enough for the purpose of the application.

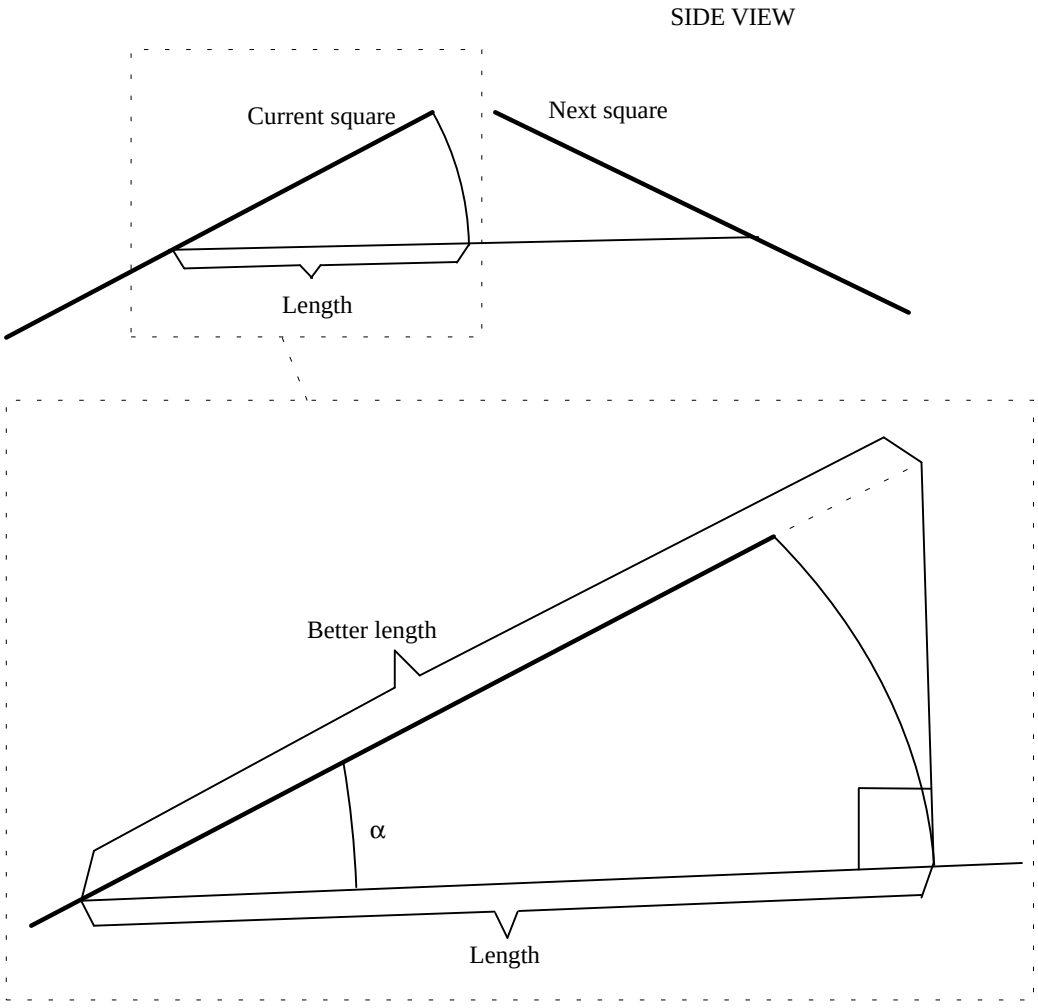

Figure 4.2: Calculating the length of the **forward** vector.

Once we have the **front** point, we need the **frontLeft** and **frontRight** points. For this we need a vector perpendicular to both the **normal** and the **forward** vectors. This is achieved by taking the cross product of these two vectors to give us the **forwardCrossNormal** vector. Since the **scales** are supposed to be squares, we decided to make the length of the **forwardCrossNormal** vector the same as the length of the **forward** vector. The point **frontRight** is found by applying the **forwardCrossNormal** vector to the **front** point. The point **frontLeft** is found in a similar way, but this time we use the inverse of the **forwardCrossNormal** vector. Using the inverse of the **forward** vector, we can find the **Back** point. The **backLeft** and **backRight** points are found in a similar fashion to the **frontLeft** and **frontRight** points.

4.1.2. Implementation issues

To do these geometric calculations, we decided to create our own libraries.
The following data structures and functions have been implemented, with emphasis placed on the efficiency of the representation.

Data structures:

The **point** only needs to have its 3-D components **x**, **y**, and **z**.
The **vector** has its 3-D components **x**, **y**, and **z**, and we have also included a field for the **length**. This field is not always required and it is more efficient to calculate the length of a vector only when required. It would however be handy field to have if the library was to be ported to an object oriented design since the length is definitely an attribute of the vector.

The **plane** includes the **normal vector**, a **point** on the plane, the equation parameters of the plane (**A**x + **B**y + **C**z + **D**) and the rotation matrix that would make the vector [0 0 1] a normal to the plane. This last field was only included later on, when it was required to make the implementation easier. Some more in-depth information about the implementation might be required here:

We decided to build a plane from each *pulse* of data. Once we have a previous plane, a current plane and a next plane, we can then start building the object with which we will represent the surface, in our case a square. This way, we "concentrate" almost all the information in three planes that can be passed as parameters to the square-building procedure. The problem is that this way we can not pass the information about the **forward** vector. This would mean passing either the whole data provided by the Polhemus tracker or at least the rotation matrix. We found that incorporating the rotation matrix to the plane made this whole problem more modular: although we are still passing the rotation matrix, we are doing so in a more subtle and elegant way.

The next step is to create a data structure for the square itself. The **square** includes a **plane**, a **forward** vector and a **normalCrossForward** vector. It also has the four **points**: **frontLeft**, **frontRight**, **backLeft**, and **backRight**.

Functions:

We need functions to:

- **normalise** a vector.
- calculate the **length** of a vector.
- **invert** a vector.
- calculate the **distance** between two points
- calculate the co-ordinates of a point after applying a vector to another point
- multiply a vector by a scalar.
- get the **cross product** of two vectors.

- compare two planes and tell if they are **similar** according to some criteria. If the two planes have their centres separated by less than 1.0 unit and the angle between their normals is smaller than 2.56 degrees, the planes are assumed to be the same plane and hence, the second plane is not processed. Variation in the degree of precision is thus allowed. These values are determined experimentally, and the user can determine his own preferred values.

We also need a data structure to represent the collection of squares (i.e. the model). The Object Format File (OFF) is used. We need a data structure to represent the OFF in memory, and two functions that will write the OFF into memory and to a file.

The data structure to hold the squares in memory has the format of figure 4.3. It consists of a header record that has information about the number of vertices, faces and edges in an array, and four links to two linked queues.

The first linked queue consists of records that hold information about the co-ordinates of a specific vertex, and a link that links the current vertex to the next. The header has a link pointing to the head of the queue, and another link pointing to the tail of the queue. The link pointing to the tail of the queue is not strictly necessary since one can find the last element by starting at the head of the queue and "going down" the queue until the last item is found. This is however quite inefficient when the queue grows bigger. The pointer to the tail of the queue provides efficiency when the program needs to add the next vertex.

The second linked queue consists of a similar record to the first queue. Instead of storing information about the vertices, it stores the information about how many vertices are required to form the face, and which vertices they are. The selection of the vertices is done by storing their position in the OFF file. As an example, a pentagon face could have its information as follows: 5 23 25 24 22 27. This would be because a pentagon has five edges, and this specific pentagon has edges

between the 23rd and 25th vertex, the 25th and 24th vertex, the 24th and 22nd vertex, the 22nd and 27th vertex, and the 27th and the 23rd vertex. Again, links pointing to the head and the tail of the queue are provided.

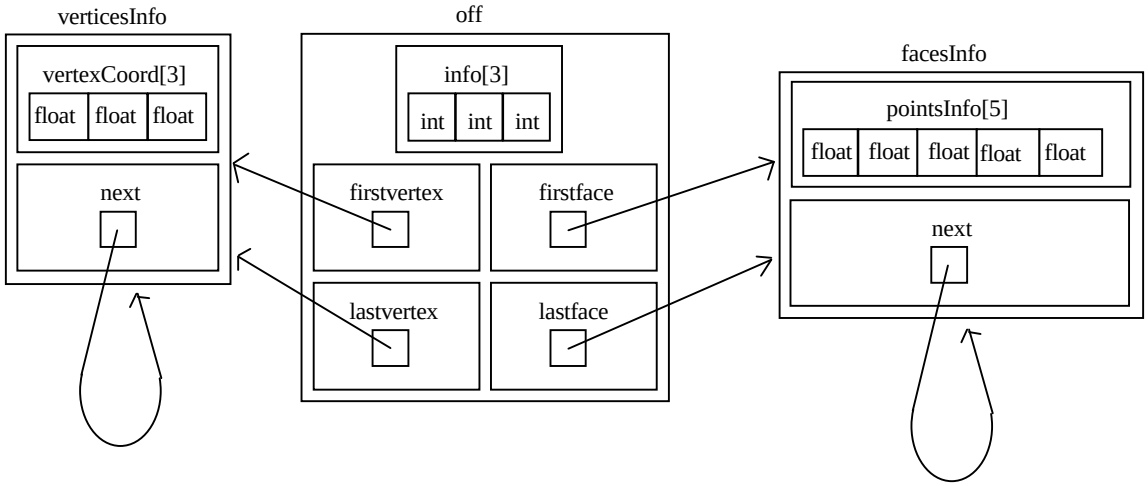

Figure 4.3: Data structure to hold the OFF data in memory

4.1.3. Conclusion

Refer to figures A13 and A14 in Appendix A.

The **scale** metaphor has given us a tool with which to map real objects into a 3-D graphics format. However, the tool is not finished yet and further investigation will continue to provide a better tool.

There are various advantages to this tool. Although the squares do not create a closed object, adding enough **scales** to the model will result in the appearance of

the model being almost completely closed. A large number of **scales** means that the display engine will lose speed. This is a negative aspect in VR because it creates latency, which hampers interactivity. However, the **scales** tool allows easy visualization of the modelled object even when the object is mapped with a low number of squares.

The orientation of the squares is dependent on the use of the tracker by the user. If the user keeps the tracker constantly oriented in one direction, the result will be ordered tile-like squares. If the orientation is not constantly maintained, the squares will not have a regular orientation. This, however, has only aesthetic consequences.

The size of the squares depends on the acceleration on the tracker. This is also user related. If the user does not maintain a constant acceleration, the size of the squares will vary. The difference in size is, however, not very apparent. Again, this will only have aesthetic consequences in the resulting model.

Since the squares are not connected, there is a feeling of texture missing in the model.

4.2. Intersecting planes

The squares from the **scales** metaphor actually depict surfaces of the object. With this in mind, we decided to investigate the possibility of having the planes representing the surfaces intersect each other, and only keep those portions of the planes that are part of the object. This sounds like a very effective method. Some examination soon shows that the method has quite a large number of drawbacks. Firstly, one can only map objects that are convex. This is similar to the **gift**

wrapping method referred to by [Preparata and Shamos 1988]. An example of this can be appreciated in figure 4.4 A. This problem can be solved if we limit the "cut" of the plane to the **previous** and **next** square as shown in figure 4.4 B. The part of the **current** plane that does not contain the **centre** point can then be discarded to give satisfactory results.

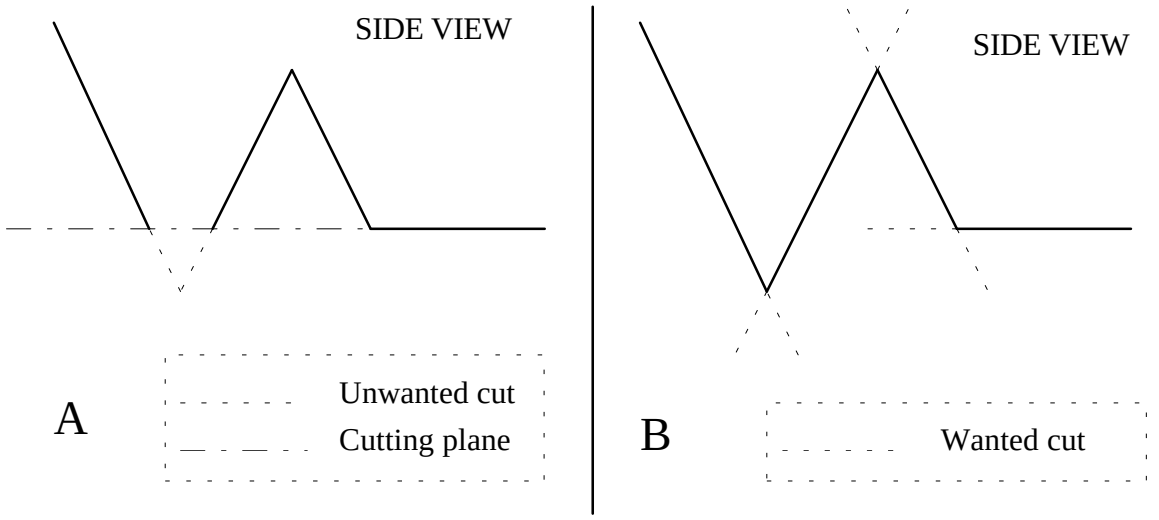

Figure 4.4: A: Cutting a non convex set of planes. B: Cutting only sequential planes.

But what about the adjacent squares (like the ones to the left or the right of the **current** square)? We know about the **next** and **previous** squares because of the sequence in which the data was captured. However, the sequence does not give any spatial information that would allow us to find out with ease which other squares are adjacent to the **current** square.

Another problem is that the intersection between two planes is not always a clean line almost perpendicular to the **direction** vector. Such an intersection would be ideal. But, this algorithm does not produce satisfactory results in all cases, as shown in figure 4.5.

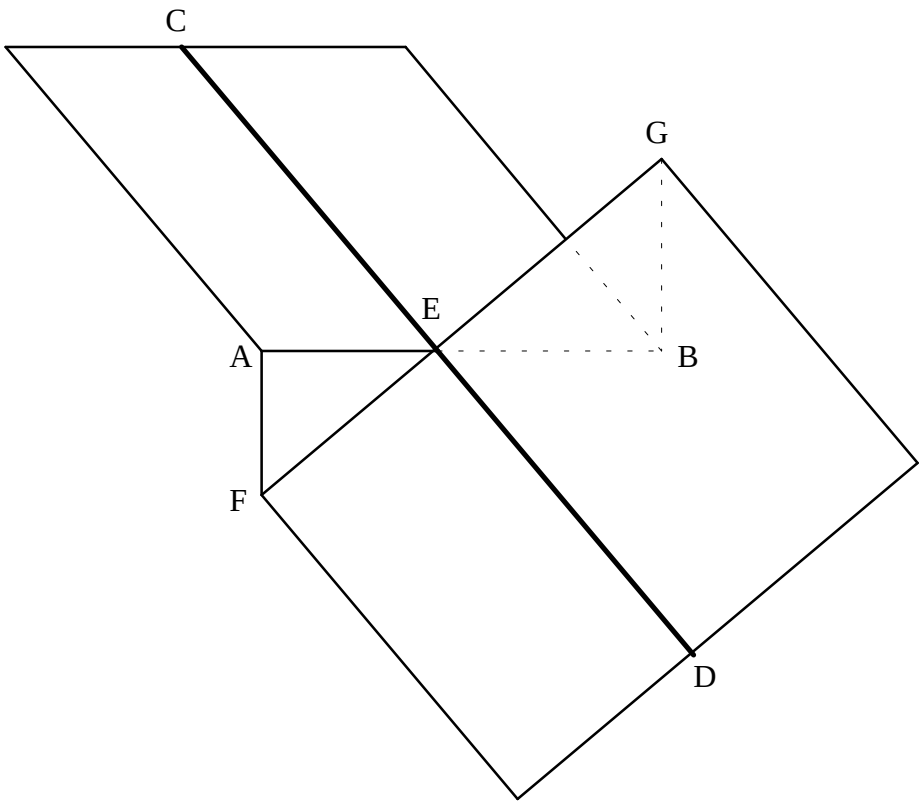

Figure 4.5: Path of the tracker over two oblique planes

Assuming that in figure 4.5 the tracker follows the path of line C-D, we can have more than one event:

- If the planes are just tilted slightly due to a small error, one would want to cut the planes along the line A-B.
- If the tilting is due to the model's topology, one would want to cut along the line C-D and create the planes AEF and BEG.

These ambiguities cause the algorithm to be non-intuitive and complicated.
For this reason, the method was discarded and we decided to look at a new approach.

4.3. The triangles

After failing to generate a simple method to create surface models with intersecting planes, we decided to use triangles instead of squares. The points in these triangles can then be displaced without the triangle loosing its planar properties.

4.3.1. Creating the triangle

Triangle creation is based on the same principle as the square. The **frontLeft** and **frontRight** points are deleted to leave a triangle (figure 4.6).

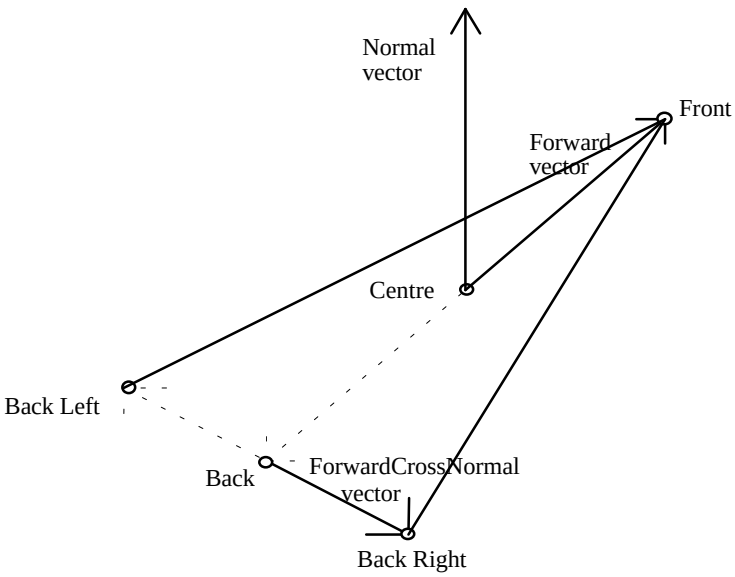

Figure 4.6: Creating a triangle

The effects of representing a surface with triangles are shown in figure 4.7.

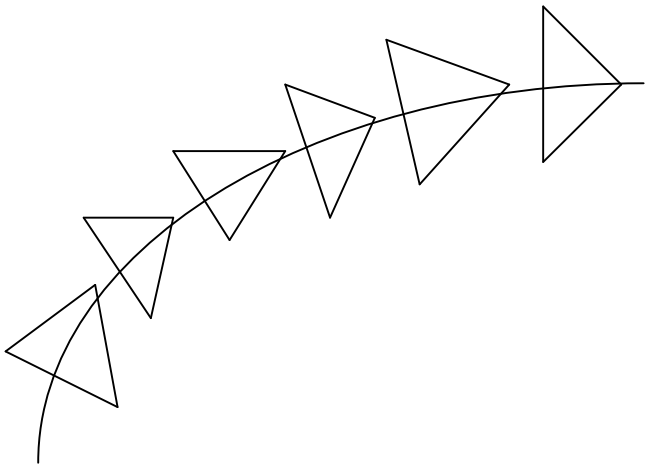

Figure 4.7: Tracing a surface with triangles

We then decided to invert the direction of every second triangle. The result is similar to figure 4.8.

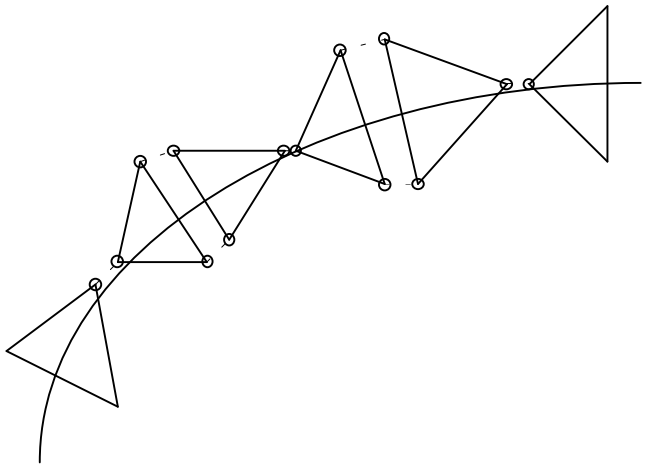

Figure 4.8: Tracing a surface with triangles in alternating directions

One can clearly see that the distance between adjacent points is very small. We then decided to merge adjacent points so as to connect successive triangles together. This is shown in figure 4.9. The merging of the points is done so that the resulting point will be midway between the original points, giving an equal share of importance to both points.

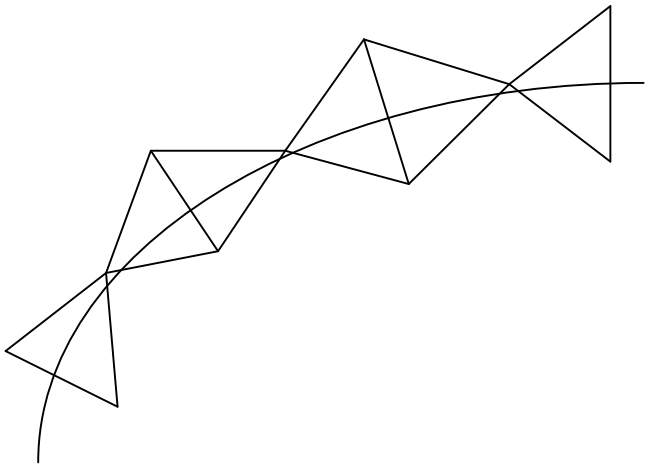

Figure 4.9: Tracing a surface with merged triangles

4.3.2. Implementation issues

The introduction of a triangle to replace the square means that new data structures are required to hold the triangle in memory. We also need to write new functions to write the OFF data into memory as well as to a file.

The data structure is very similar to the one in figure 4.3 except that the array **pointsInfo** in **facesInfo** needs to hold one less field. This array contains the information about how many and which points form a specific face. The triangle has one less point than the square.

Apart from this, only a function to **merge** the triangles together is needed.

4.4. The "toilet paper" metaphor

4.4.1. Creating the "toilet paper"

As stated before, triangles are not optimal in visualizing a surface, so we have decided to transform these triangles into quadrilateral polygons. The problem is that a quadrilateral polygon will lose the properties of the triangle of always being a planar polygon. We then decided to make this quadrilateral polygon out of triangles by connecting successive side points together as shown in figure 4.10.

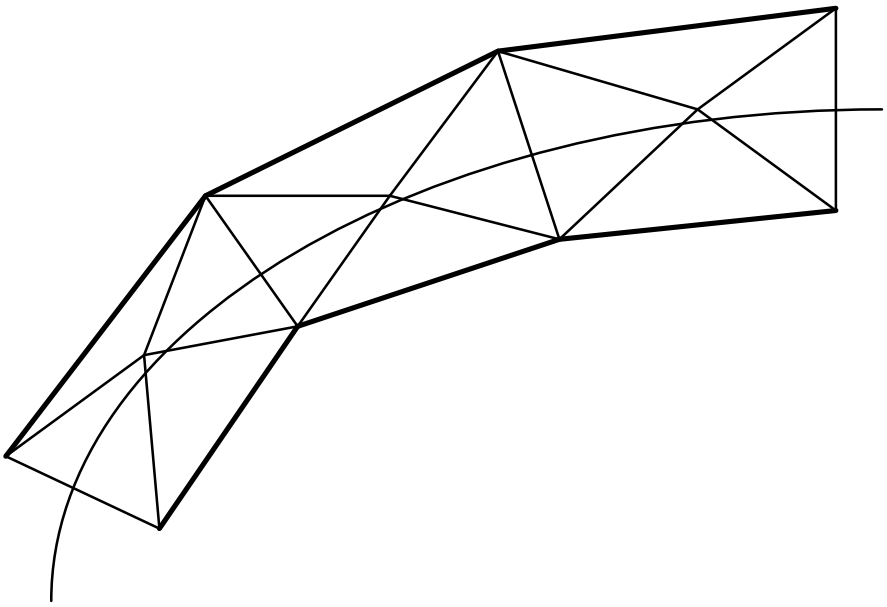

Figure 4.10: The **toilet paper** metaphor.

Joining the triangles makes them look like a bandage. It gives the feeling of "wrapping" or "mummifying" the object with **toilet paper**, hence the name for the metaphor.

4.4.2. Problems and solutions

The **toilet paper** method is obviously different to the **scales** method. These differences manifest themselves as various problems, not only visually but also algorithmically.

4.4.2.1. The **forward** vector versus the **direction** vector

Until now, we have used the **forward** vector to create the triangles. When mapping an object, we tried to keep the **forward** vector as close to the direction of the tracker as possible. Indeed, if the **forward** vector has an awkward direction, horrible things can happen, as can be seen in figure 4.11 and 4.12.

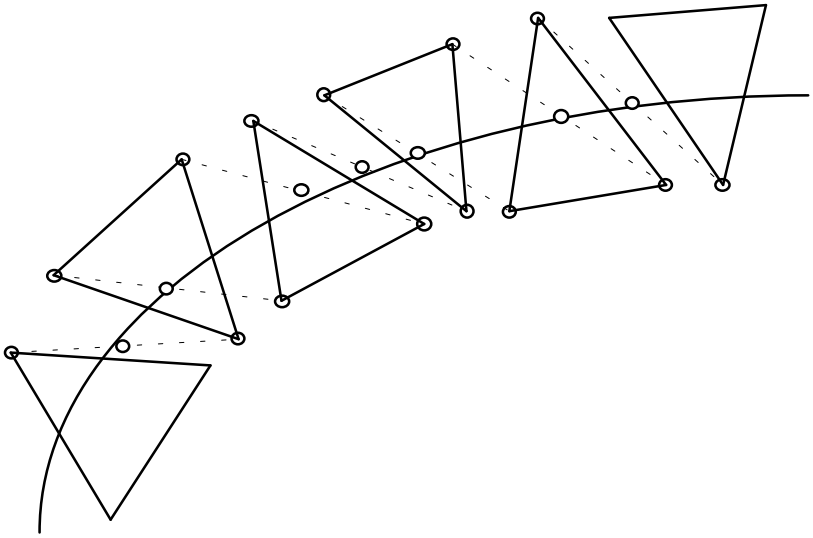

Figure 4.11: Triangles before being merged with the **forward** vector perpendicular to the path of the tracker and showing the position of the resulting merging points

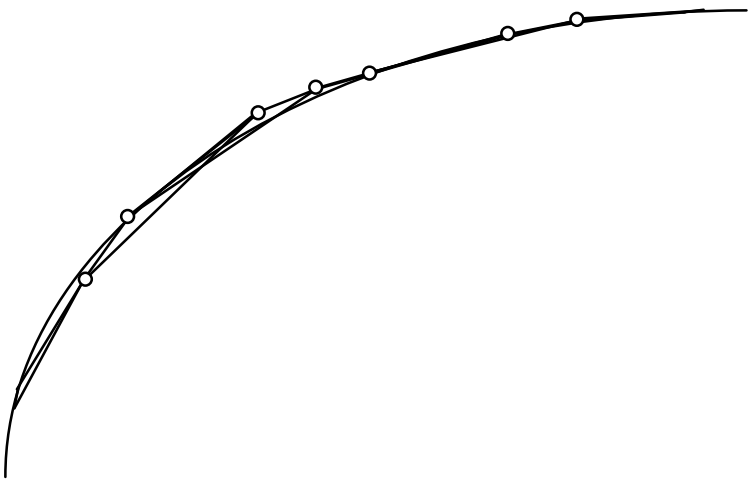

Figure 4.12: Triangles with **forward** vectors perpendicular to the path of the tracker, after merging

As can be seen, the results are catastrophic. These can be avoided by using the **direction** vector instead of the **forward** vector. These problems arose because, by joining the triangles the way we have, we have implicitly represented information about the path of the tracker. This path corresponds to the **direction** vector.

The use of the **direction** vector instead of the **forward** vector means that the direction in which the tracker is pointing is irrelevant. This makes the program easier to use as one need not worry about whether or not the tracker is pointing in the correct direction.

4.4.2.2. The width of the toilet paper

In the **scales** metaphor, we set the length of the **forwardCrossNormal** vector to the same length as the one for the **forward** vector. This was so that the **scales** would be square. The effect of this is that an increase in the distance between the **previous** and **current** squares, or between the **current** and **next** squares will result in an increase in the size of the square, and vice-versa. For the **toilet paper** metaphor, this means that the thickness will vary according to the velocity of the tracker. We decided that this effect is not appropriate and that a constant width would better suit the metaphor (after all, real toilet paper does have a constant width). We also decided to make that width representative of the real width of the tracker. We decided to find this width experimentally (i.e. through trial and error). To achieve this, we did the following experiment: We traced the horizontal surface of a "floor-wall" intersection with the tracker right up against the wall. We then traced the wall with the tracker right up against the floor, as can be seen in figure 4.13. This gives us a result similar to that of figure 4.14.

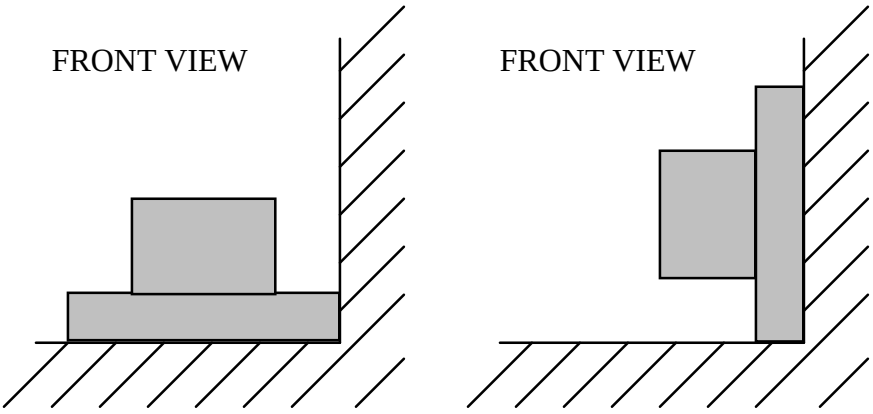

Figure 4.13: Running the tracker on the "wall" and the "floor" corner

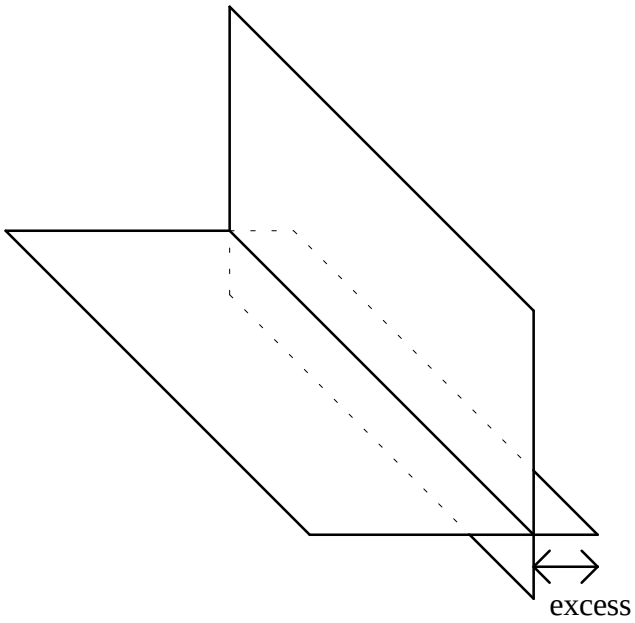

Figure 4.14: The resulting surfaces from the experiment in figure 4.13

The width of the **toilet paper** is modified until the two surfaces in figure 4.14 meet perfectly with each other, with the excess being zero. This will result in the width of the surface corresponding to the width of the real tracker.

4.4.2.3. The real **centre** of the Polhemus tracker

The previous statement is, however, only true in the special case where the **centre** point corresponds with a point at the bottom of the tracker. If the **centre** point corresponds with a point that is higher than the bottom surface of the tracker, the width would be smaller than required, as seen in figure 4.15.

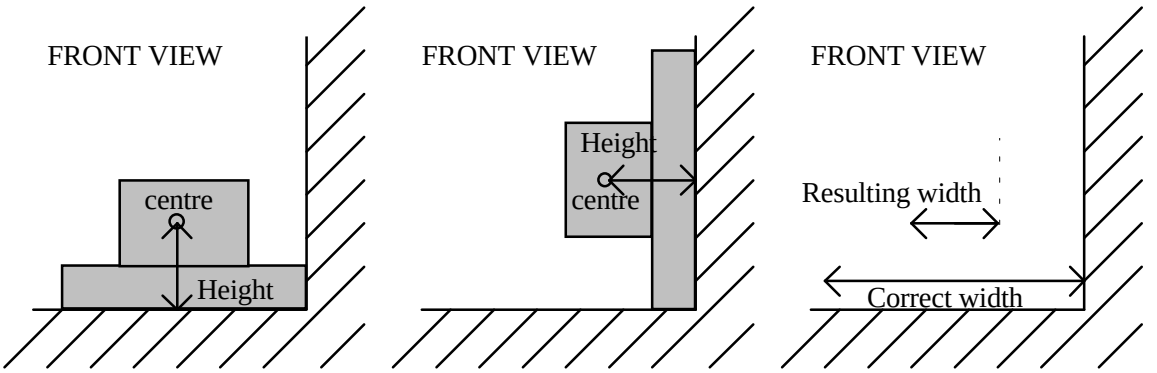

Figure 4.15: Resulting width if the **centre** is at a certain height

This is the case with the Polhemus tracker. This information is not given in [Polhemus 3Space 1993]. To solve this, we did the following:

The previous experiment gives us the minimum width that would be encountered if the two surfaces being mapped were perpendicular along the path of the tracker. The following experiment gives us the maximum width with the two surfaces perpendicular to each other along the path of the tracker.

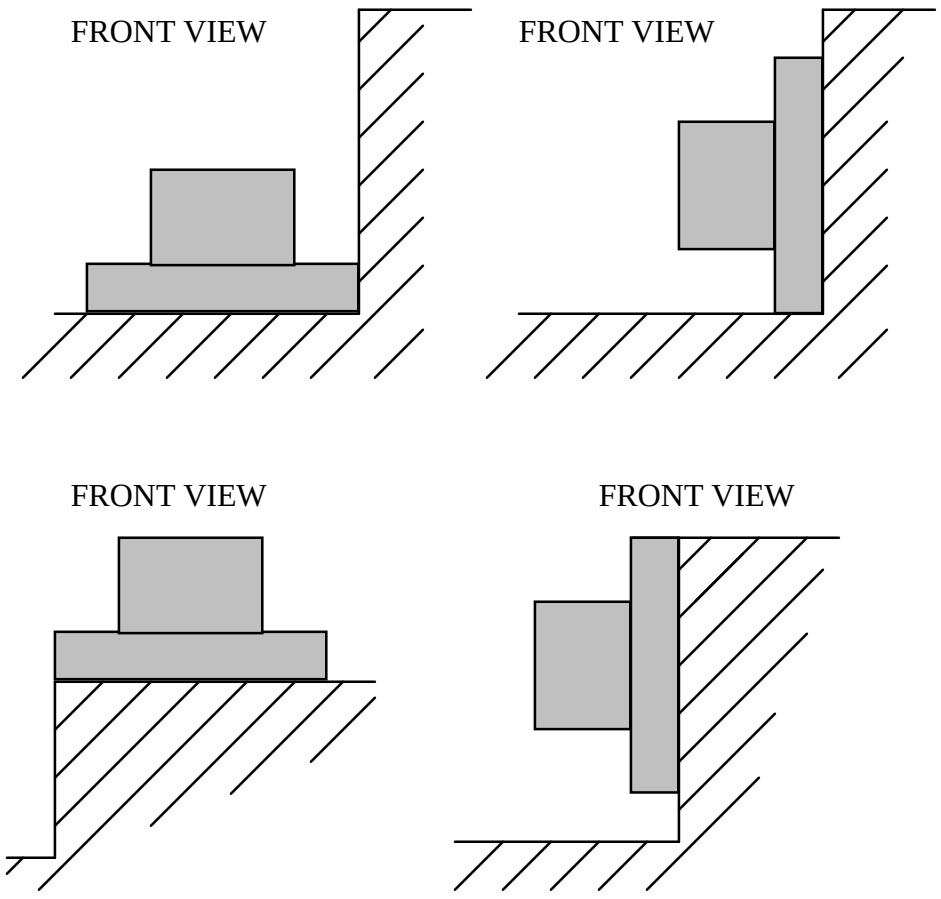

Figure 4.16: Running the tracker along the "floor", the "wall" and the "roof"

The maximum width is found when the gap in figure 4.17 is zero, even if the excess in this case is large.

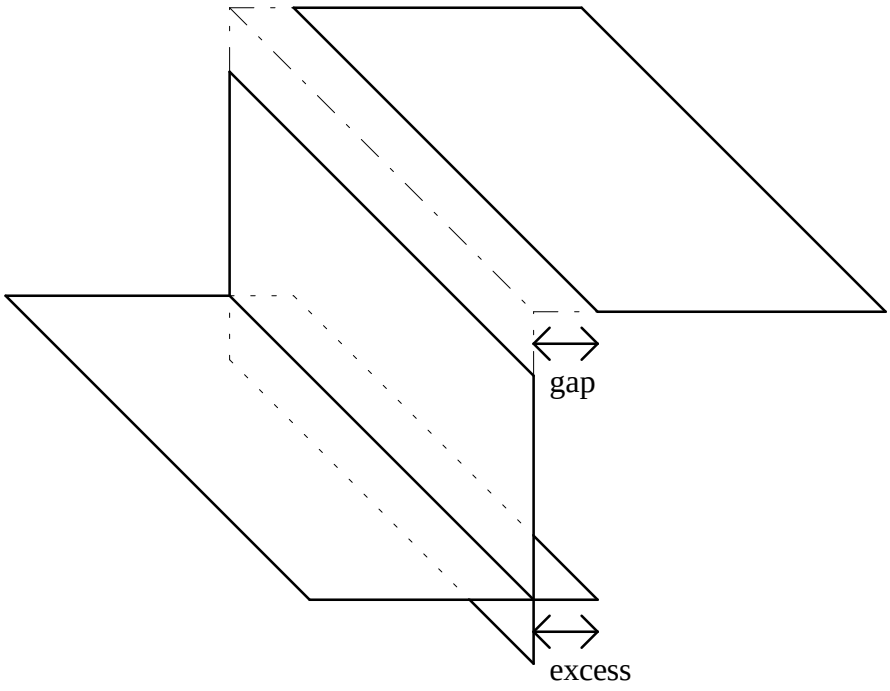

Figure 4.17: The resulting surfaces from the experiment in figure 4.16

The correct width is found when the gap is equal to the excess; In other words, the average between the maximum and minimum widths.

With the maximum and minimum widths, we can find the height at which the **centre** given by the Polhemus tracker is located (see figure 4.18).

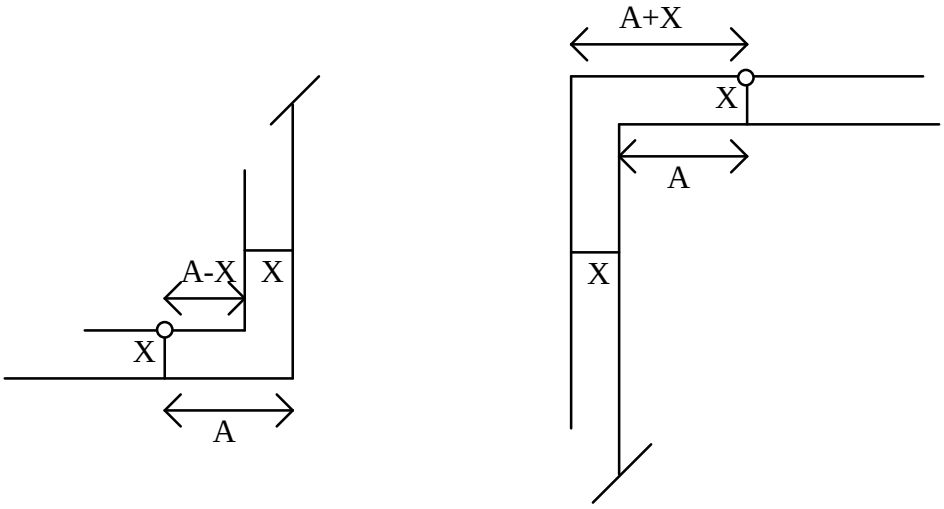

Figure 4.18: Deriving the height of the **centre** point

Where:

- maximum width = 2*(A+X).
- minimum width = 2*(A-X).
- correct width = 2*(A).

X = (correct width - minimum width)/2.

X = (maximum width - correct width)/2.

X = (maximum width - minimum width)/4.

Using X, we can find the correct **centre** as follows: Use the **scalarMultiplication** function on the **inverse** of the **normal** vector. With this new vector and the

incorrect **centre**, we can use the **pointFromVectorAndPoint** function to get the correct **centre**.

4.4.3. Implementation issues

The only new implementation required is the addition of two new functions to put the new triangles into memory and to write them to a file, in OFF format.

A way of increasing and decreasing the width of the **toilet paper** has also been added by making the width variable.

We have also added a way to pause and to resume the creation of the **toilet paper**.

4.4.4. Conclusion

The **toilet paper** metaphor has given us a second tool with which to map real objects into virtual 3-D objects. As with the **scales** metaphor, the **toilet paper** metaphor does not create closed objects, although adding enough **toilet paper** bands gives the appearance of a closed object. The advantage of the **toilet paper** over the **scales** is that, because the **toilet paper** is linked together, it has a feeling of texture. This allows the object to be visualized more easily when a reduced number of polygons are used. The other advantage is that, because there is no need for the **forward** vector, the user does not need to consider the orientation of the tracker, making the tool more user-friendly.

4.5 Conclusion.

In this chapter, we have achieved the creation of two tools that allow us to map a real object into a surface model of that object. These tools are intuitive and computationally efficient. Part of the user friendly interface is due to the choice of a VR system to drive the application.

4.5.1. Disadvantages

The major problem with this tool is that generated objects are not closed objects. This is not a major concern since, as stated before, the aim of the project is to generate a tool that will create surface models and not closed solid models. Another disadvantage arises from the approximation, especially in the **toilet paper** method. As an example, consider figure 4.5. The result using the **toilet paper** tool would be as in figure 4.19.

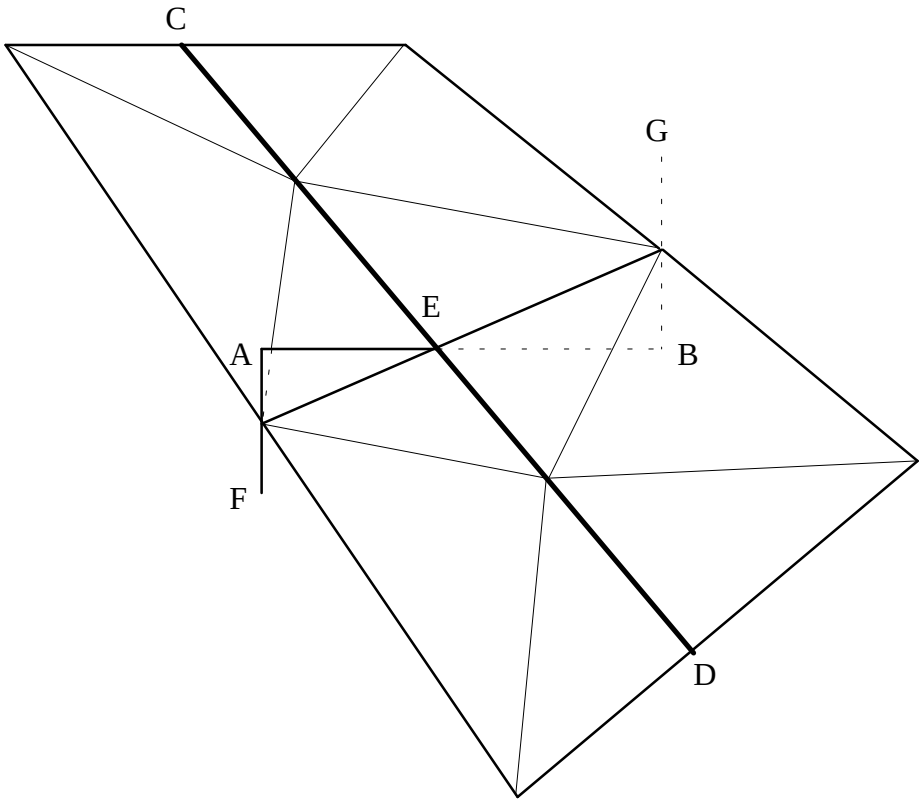

Figure 4.19: Approximation to the surfaces in figure 4.5 using the **toilet paper** tool

As one can see, the surfaces have been approximated. As a result of this approximation, the surfaces AEF and BEG have disappeared.
As stated earlier, we do not require a great deal of precision, since our major concern is to achieve a general shape of the object.
Other disadvantages are caused by the Polhemus tracker itself: short range, non-metallic objects only. These are not our concern.

4.5.2. Extensions

Possible extensions to these tools would include the use of two trackers. At the moment, the object has to be fixed in the same position. The second tracker can be attached to the object, allowing the user to move the object freely. The second tracker would become the origin of the first tracker. This can be easily achieved by subtracting the position of the second tracker from that of the first tracker. This would correspond to a vector subtraction as seen in figure 4.20.

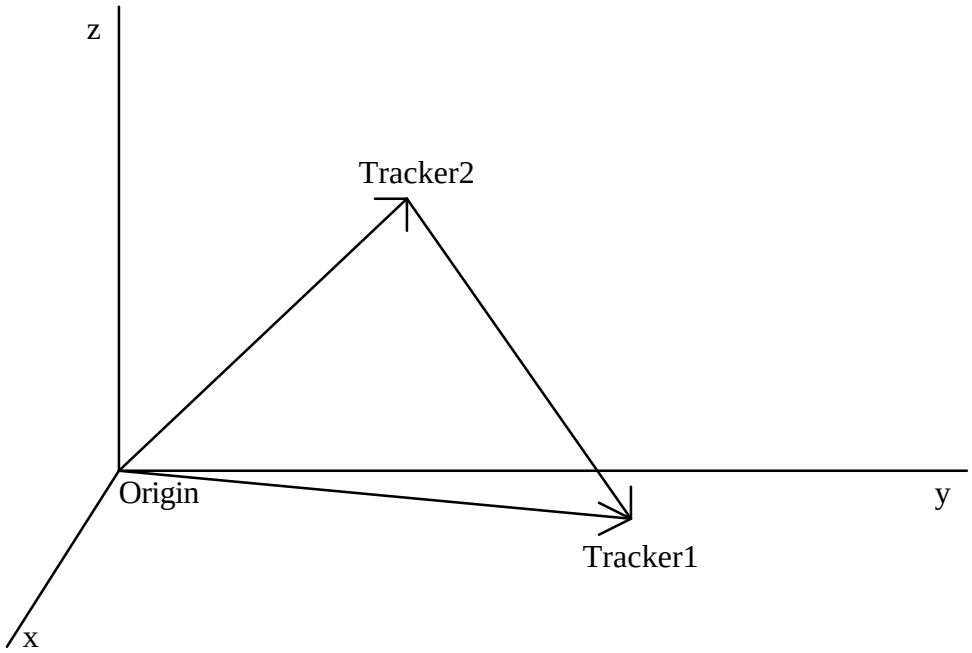

Figure 4.20: Position of the first tracker with respect to the second tracker

The vector **Tracker2Tracker1**, which gives the position of the first tracker with respect to the second one, is equal to vector **OriginTracker1** (the position of the

first tracker with respect to the transmitter) minus vector **OriginTracker2** (the position of the second tracker with respect to the transmitter).
Similarly, one would only need to multiply the rotation matrix of the second tracker by that of the first tracker to obtain the orientation of the first tracker with respect to that of the second tracker.

Other extensions might include providing a reference marker to designate the position of the tracker without actually mapping that information into the OFF file. At the moment, the only way the user has to know where the position of the tracker is, is by the creation of one of the end of the **toilet paper**. A marker would increase the ease of use of the tool. This would only involve displaying an arrow-like object (a cone or a pyramid would do just fine) at the position of the tracker and with the **direction** or **forward** vector as orientation.
Another extension might be a facility to erase parts of the **toilet paper**. This could easily be done by using a stack to store the **toilet paper** band, and popping pieces of the **toilet paper** band as requested by the user.

All these improvements are mainly concerned with the user interface, which is beyond the scope of this thesis.

As stated earlier, the results of this last chapters are two computationally efficient tools that allows a user to map a real life object into a surface model. These tools were implemented under a VR system. This VR system and the incorporation of the tools into this system are discussed in the next chapter.

Chapter 5

RhoVeR: The Rhodes Virtual Reality System

RhoVeR, the Rhodes University Virtual Reality system, was created by members of the Rhodes University Virtual Reality Special Interest Group (VRSIG) to cater for various needs. These needs are:

- To investigate aspects related to VR systems. Such aspects include using a parallel/distributed VR system to address problems found in previous VR systems, such as latency and general performance.
- To serve as a standard platform for research involving the creation of virtual environments at Rhodes University.

In this chapter, we will describe the RhoVeR system. This is closely based on [Bangay *et al* 1996a], [Bangay *et al* 1996b]. We will also describe its use in building our application.

5.1. Description

RhoVeR is a fully parallel/distributed VR system. It can be run across a network of machines with different architectures, or on a single machine supporting concurrent processing.

RhoVeR can be thought of as being an operating system in which VR applications can be built. For that purpose, it provides the user with an environment and tools to support the creation of VR applications.

RhoVeR has been designed with a high degree of modularity. This makes the environment highly configurable to suit any application, thus allowing the creation of a wide range of applications. This modularity also means that the incorporation of new VR equipment can be done easily, without affecting the equipment already present.

The design of RhoVeR has also been influenced by VR aspects such as latency and VR entities (like worlds and objects).

5.2. Structure

A typical RhoVeR application consists of a collection of processes running concurrently. Each process is an instance of a module. These modules correspond to the components of the VR system.

5.2.1. Communication

There are three mechanisms available in RhoVeR to allow communication between instances of modules:

- An event-passing facility allows point-to-point communication between processes by sending messages (called *events*). The types of events are pre-defined, and all processes have particular ways of responding to specific events. The response usually changes the behaviour of that process.

- A virtual shared memory (VSM) database. This is a database of information concerning the object processes. In theory, each process should have a copy of the database. However, for efficiency purpose, a single copy is kept on each machine running the VR system. Any process is allowed to access data from the database. However, only the process that owns the data, or a process that owns that process (i.e. its parent or owner), can alter that data. Only the data that is of general interest to the objects, or the world, is kept in this virtual shared memory, i.e. data such as the position and orientation of an object.

- A shared table. The data that is of interest primarily to a single object process only is kept in a shared table (called the ShapeData table). This table is cached locally. An example of such data would be the shape (the polygon-mesh) of an object. This is usually only of interest to that object. If another object is interested in that data, it will be sent on demand, using event-passing mechanisms.

5.2.2. Types of modules

There are various types of modules. These include world, object, Input/Output (I/O), application support, and virtual shared memory (VSM) manager modules [Bangay *et al* 1996b].

- The **world** module is responsible for the control of the modules in that world. This allows the creation of various independent worlds which can run concurrently. The **world** module is also responsible for the control of global attributes, such as gravity.

- The **object** module is responsible for the behaviour and appearance of a 3-D object. The shape of the object is loaded from an OFF file and is kept in the ShapeData table of that object. The position and orientation of the object are kept in the VSM.

- The **I/O** module is responsible for the control of I/O devices. Input devices include the Polhemus InsideTrak and the Fifth Dimension Technologies 5th-

Glove devices. RhoVeR uses the MESA rendering engine as the output device, to keep compatibility with the OpenGL standard.

- The **application support** modules support specific applications. An example of this would be the timer module, used in the *ant psychology* application [Bangay *et al* 1996b].

- The **VSM manager** module is responsible for controlling the access of processes to the VSM database. Only owner or parent processes are allowed to alter the information in the database. The **VSM manager** is also responsible for broadcasting the database to other machines when a change has occurred.

To join all these modules together, RhoVeR provides *support libraries*. These *libraries* include:

- The **control** library manages the parenthood and ownership of objects. The parenthood stores the position and orientation of the child object relative to the parent object. The ownership allows the owner object to change the information of the owned object. Thus, if object A is the parent of, and owns object B, a change in position of object A will cause the corresponding change in position of object B. This is used as an example in the *ant psychology* where the body of the ant is the parent and owner of its legs. When the body moves, the legs move accordingly as if they were glued to the body.

- The **event** library manages the message-passing. This includes sending messages between different machines.

- The **ShapeData** library manages the information kept in the ShapeData tables, and ensures that requests for that information are sent to the requesting party.

The **process management** libraries control the start-up and termination of processes. They also control the merging of various similar processes into one single process. Again, consider the *ant psychology* application where various ants are wandering around the world. These ants have the same behaviour and, apart from particular attributes like "race" and colour, they have the same characteristics. These ant processes can then be merged into one single process "controlled by simple multithreading, and consequently (reducing) the load of active processes" [Bangay *et al* 1996b]. This merging does not happen automatically, and has to be done by the merging process.

5.3. The **scales**/**toilet paper** application

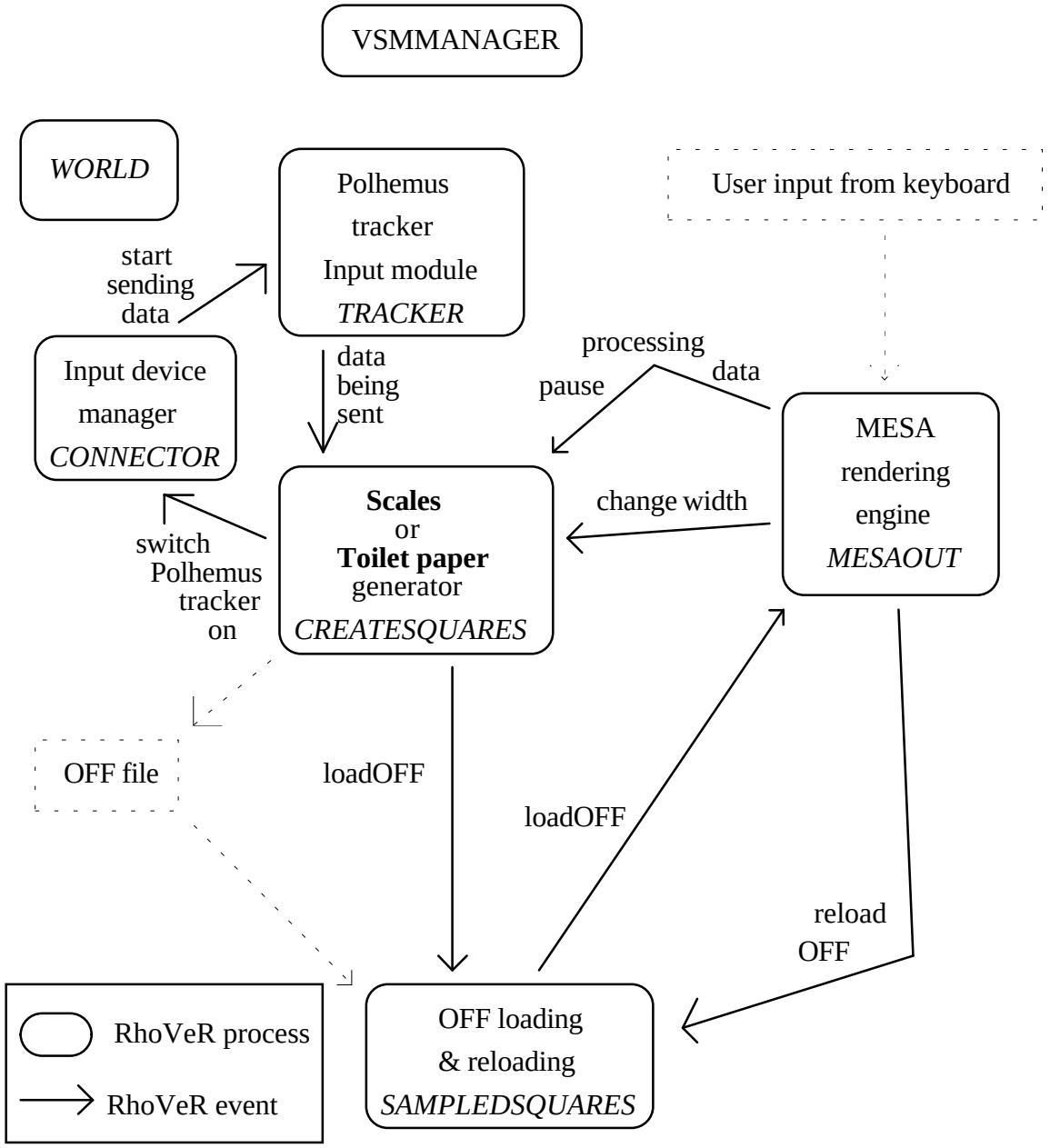

Figure 5.1: Outline of the **scales** or **toilet paper** application running under RhoVeR.

The RhoVeR system starts by creating the **world** and **VSM manager** processes. (Refer to figure 5.1 for more information. The names printed in *italics* in this chapter refer to names used in the code itself).

The **VSM manager** "spawns" instances of the following modules:

The module responsible for creating the OFF object, *CREATESQUARES*.

The module responsible for loading and reloading the OFF object from file, *SAMPLEDSQUARES*. Loading and reloading the object consists of sending an event to the MESA rendering engine.

The module responsible for the managing of the input devices, *CONNECTOR*.

The module responsible for the Polhemus tracker, *TRACKER*.

The module responsible for drawing the view of the world, *MESAOUT*.

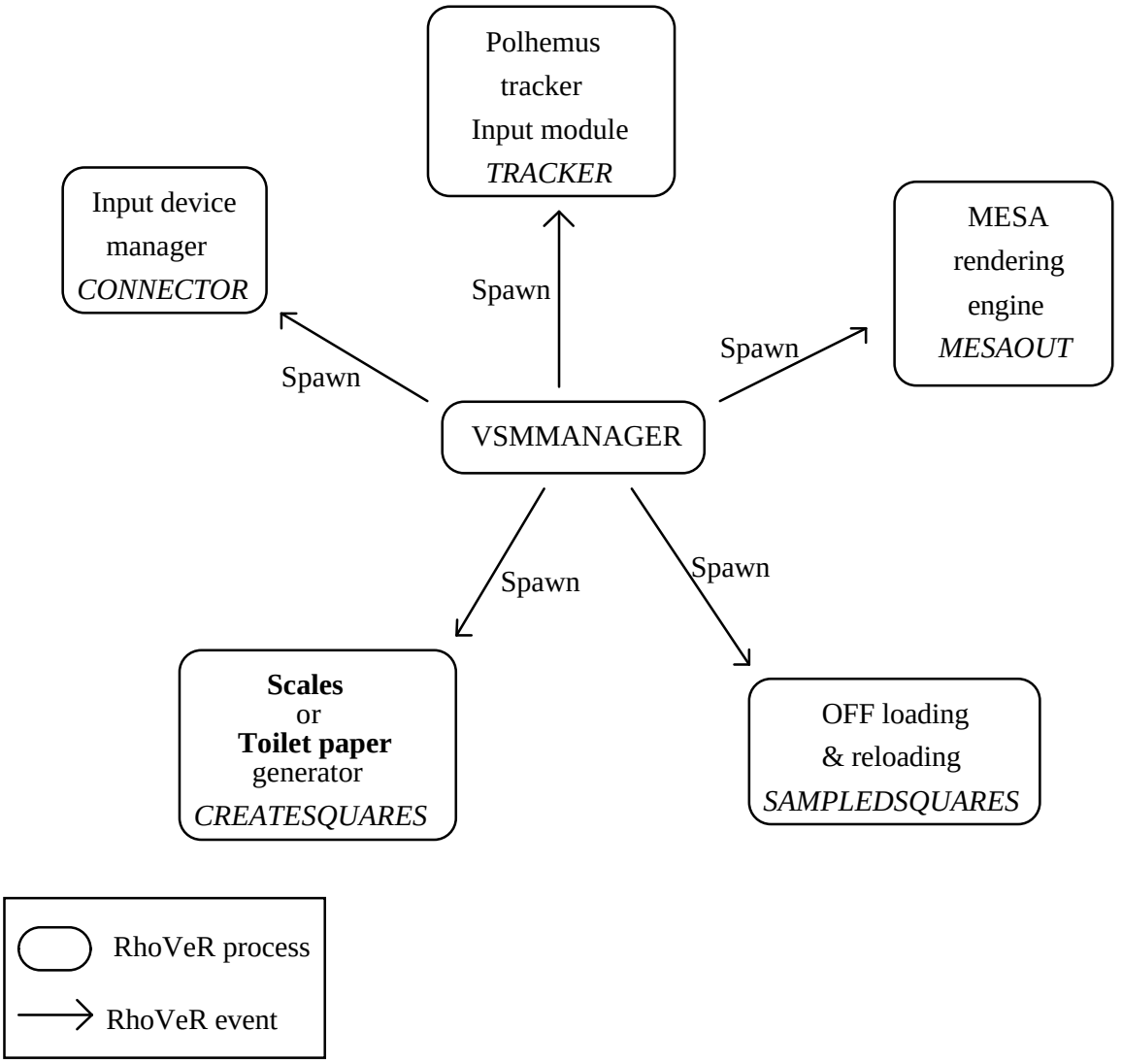

Figure 5.2: Spawning the processes.

When the process *CREATESQUARES* "spawns", the event sent is an *EVENTOBJECTINIT*. This event makes the *CREATESQUARES* event respond according to the initialisation procedure *createSquare*. This initialisation sends an

event to the *CONNECT*OR process telling it to connect to the Polhemus tracker (*EVENTCONFIGUREDEVICES*) as seen in figure 5.3. The *CONNECTOR* process sends an event (*EVENTCONNECTTODEVICE*) to the *TRACKER* process telling it to connect to *CREATESQUARES* and start sending data. This data is sent as *EVENTTRACKERDATA* events. These events are received by the *CREATESQUARES* process which handles them according to the pre-defined procedure *TrackerHandlingProc*. This procedure creates a plane from the data. If it has two previous planes, it can start creating the **scales** or the **toilet paper**.

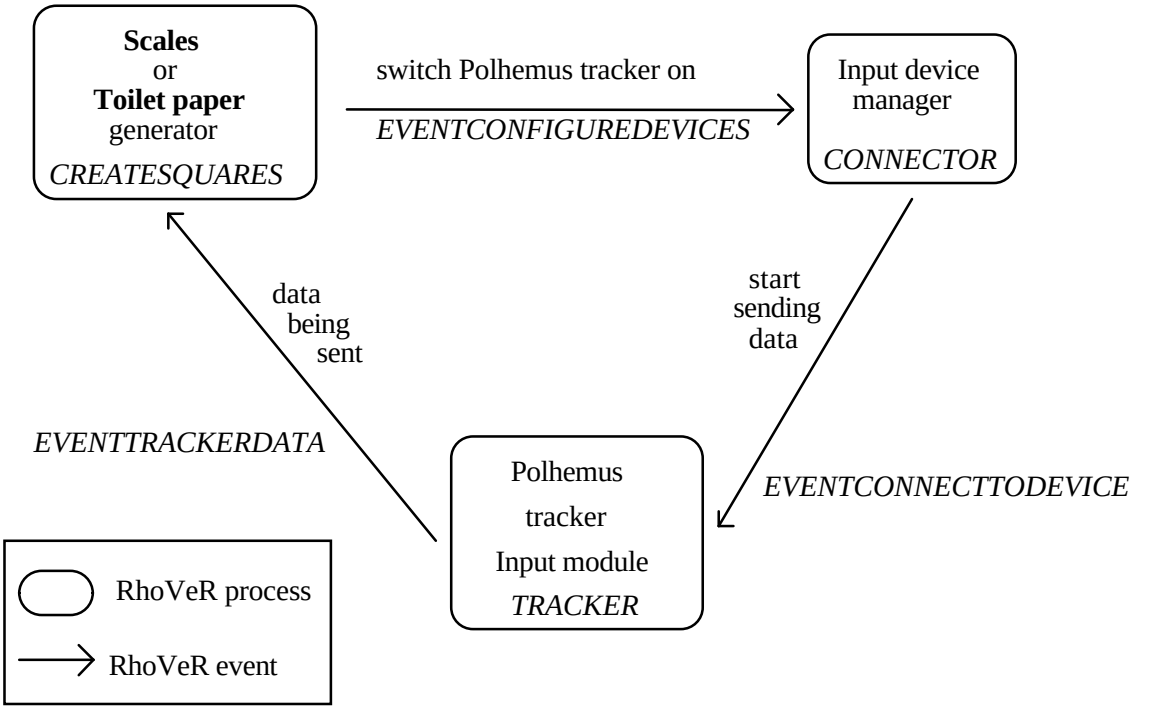

Figure 5.3: Getting the data from the Polhemus

When the *SAMPLEDSQUARES* process "spawns", its initialisation procedure sends an event (*EVENTUPDATE*) to the *MESAOUT* process to tell it to change the

position and orientation of the world view and to start displaying the world. It then sends a second event telling it to load a cube object for reference (*loadOFF*).

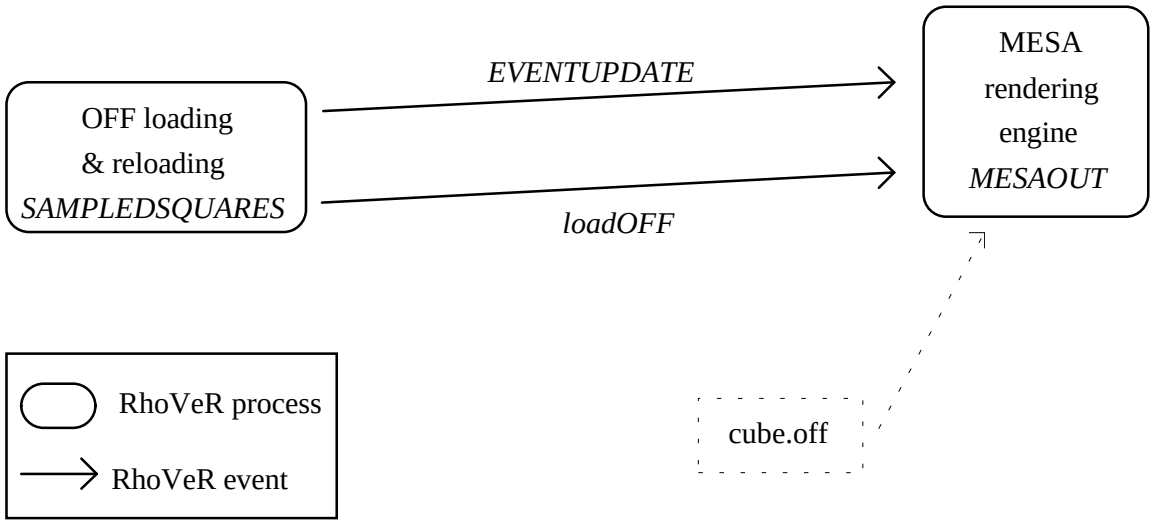

Figure 5.4: Starting the world view

Once the *CREATESQUARES* process has created the **scales** or **toilet paper**, this is dumped into an OFF file (*squares.off*). *CREATESQUARES* then sends an event to the SAMPLEDSQUARES process telling it to reload the off (*sampledReload*). This event causes the SAMPLEDSQUARES process to send an event (*loadOFF*) to the MESAOUT process to get and display the OFF file.

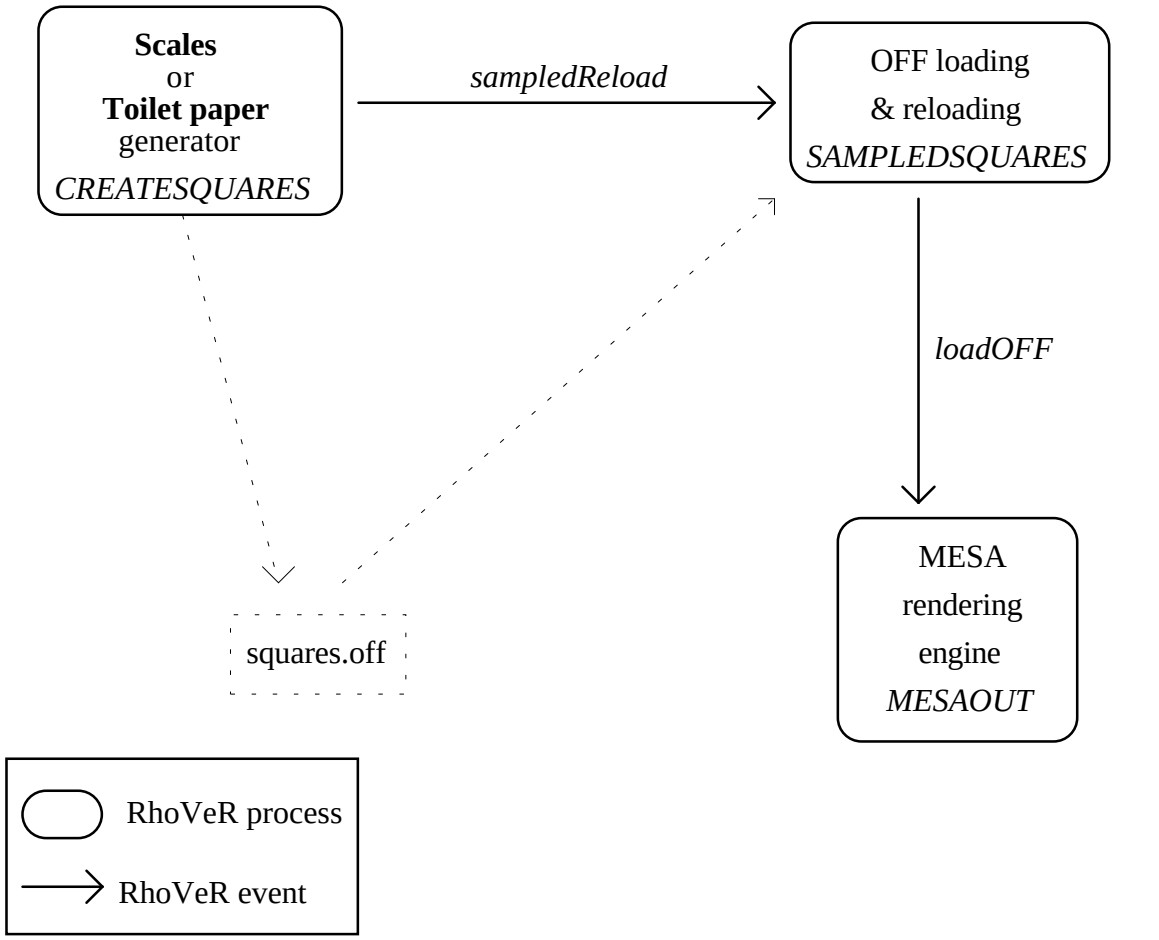

Figure 5.5: Loading the object file.

The user can give input from the keyboard to change the width of the **toilet paper** or to pause the creation of **scales** or **toilet paper**. This is done through the *MESAOUT* process. It sends an event (*EVENTENABLETRACKER*, *EVENTDOUBLEWIDTH*, *EVENTHALVEWIDTH*) to the *CREATESQUARES* process and a specific procedure (*enabledTracker*, *doublePolhemusWidth*, *halvePolhemusWidth*) handles the event.

The user can also reload the object manually. This is also done through the *MESAOUT* process. It sends an event (EVENTRELOADOBJECT) to the *SAMPLEDSQUARES* process, which then reloads the *squares.off* file.

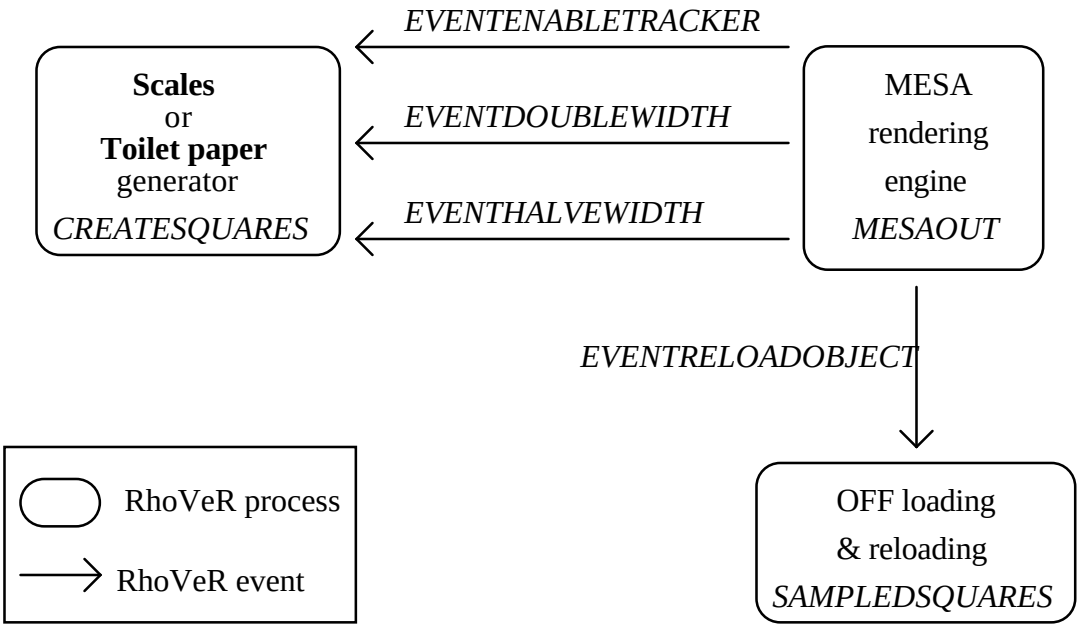

Figure 5.6: Handling user requests.

As has been shown here, creating an application in RhoVeR is not too complicated once the system is understood properly. However, the lack of proper documentation was an obstacle that could only be overcome by studying the implementation code of the RhoVeR system. Such documentation has recently been added [Bangay *et al* 1996a] [Bangay *et al* 1996b] and should be improving as the system is developed.

5.4. Further development

As it stands now, the **scales** or **toilet paper** application writes and reads the model to or from an OFF file every time a new **scale** or **toilet paper** band is created. This causes a considerable amount of overhead, affecting latency considerably, as the file becomes large. A buffer-like method in which the process waits to generate a certain number of polygons before writing the OFF file, is incorporated in the code. Although it seems to ameliorate the latency problems, it means that the data is shown in bursts at a time. This can affect the user's perception of the modelling process.

Future work might include sending the OFF data directly to the ShapeData. This should reduce the amount of I/O, and thus reduce this latency problem.

5.5. Conclusion

Considered a second generation VR system [Bangay *et al* 1996b], RhoVeR is a powerful VR engine. It is, however, in its infancy and as such, lacks extensive documentation. Nevertheless, the documentation available and the help of the Rhodes VRSIG is sufficient to create VR environments such as this one.

The consequences of using VR to develop our modelling tool is a nice and intuitive application. Since input and output devices are handled by the VR system, this allows modular separation between the various parts of the application (collecting the data, handling the geometrical information, creating the object, displaying the object). Since the various processes run concurrently, this allowed the visualization of the complexity of each process. While some processes were finished, others were still running.

Chapter 6

Conclusion

6.1. Advantages and results

6.1.1. An efficient implementation

By employing most of the data given by the Polhemus tracker in our tools, we have managed to create an application that has the following advantages over other methods:

- A simple and efficient implementation. We found simple and elegant ways to implement the modelling tools. As an example, consider the problem with Euler angles: Getting the **forward** and **normal** vectors from azimuth, elevation and roll turned out to be too complicated and computationally expensive. Using quaternions instead solved the same problem in a more efficient, more elegant manner. Similarly, using intersecting planes faced us with the problem of finding ambiguities. This was solved by approximation in the **toilet paper** metaphor.

- A real time system. Most of the modelling tools capture the data and then need to process it before any kind of displaying can take place. Our tool processes the data and displays the final object in real time. This could only be achieved by using computationally cheap procedures. These procedures are the result of the simplified approach and the selection of simple and efficient methods to represent the solid. These methods are polyhedral B-reps and surface models.

- A simple user interface. The vast variety of data results in an implementation where only one user is required to build the object. The only input devices needed are the Polhemus tracker and a keyboard for optional functions like pausing the creation of the model or changing the width of the tracker.

- A low latency within the VR system. Because of its efficient implementation, the process that creates the object has a very low latency. This needs some clarification:

  The latency problems can be divided into three sections:

  - The creation of the object. This includes the collection of data from the Polhemus tracker, its processing and the generation of the surface model.
  - The transfer of the object. This involves writing and reading the object to or from hard drive memory in the form of an OFF file.
  - The Mesa rendering engine. This involves the rendering and displaying of the object.

  At the moment, the section with the largest latency is the transfer of the object. This is mainly due to the fact that, as the number of polygons increases, an

increasingly large file has to be written and then read to or from the hard disk. It involves I/O and very slow storage memory. This process depends heavily on the number of polygons, thus the order of the algorithm seems to be of O(n). This latency problem can be improved dramatically by sending the object directly into the ShapeData shared memory table. This has the advantage of being in very fast access memory (RAM). Since the ShapeData is cached for each process, the major decrease in speed would be the overhead of possibly sending the information across a network. If the processes are on the same machine, the cost of transmitting this information is insignificant.
The next largest latency occurs with the Mesa rendering engine. As the number of polygons increases, the rendering and displaying process becomes more expensive. This process seems to be of order of at least O(n).
The creation of the object is the cheapest process of them all. This is understandable since the algorithm is only of order O(constant). On each loop, the process just gets a fixed amount of data, creates a fixed number of polygons, and adds this fixed number of polygons to a structure. This addition process does not depend on the number of polygons already added.

- A computationally cheap algorithm. As stated before, the tool for creating the surface model is very efficient. The major latency problems are with the VR system itself. Achieving effective algorithms by using richer input devices is quite relevant, as can be appreciated from the following example. Consider [Hoppe *et al* 1992]: The only input used is a set of scattered data points. From those points, a complex and expensive algorithm is used to get surface normals at those points. Although the work has merit, it would have been simpler and faster to use the Polhemus tracker to provide the points and their normals.

6.1.2. Using Virtual Reality

Implementing the system under RhoVeR means that the system has all the benefits of a VR application. Some of these benefits translate into a more intuitive tool. As an example, consider that since the user uses his hands to trace the real object with the tracker, he does not really need to be looking at the real model the whole time. He should be able to operate the tracker while looking at a virtual picture of the object being mapped. His hand-eye co-ordination should be as good as when looking at the real object. The user can use a HMD to visualize the virtual object. However, sometimes the user does need to look at the real object, since at the beginning, he cannot see the virtual object since he has not mapped it yet. This would mean providing a facility that will allow visualization of the real and virtual objects at the same time, while the user is capturing its topology. This is done by using a HMD that provides a semi-transparent visor through which the user can see the real world.

6.2. Problems

As with any real system, my application is not flawless. The main problems involve the limitations of the Polhemus tracker as reviewed in Chapter 3. The size of the tracker itself can also cause problems when mapping objects that have sharp turns and small features. Mapping such parts can be tricky with the tracker not adjusting itself to the features properly. An example of this can be perceived in figures A7 and A8 in Appendix A, where we tried to map a face.
A limitation of the **toilet paper** application is that it approximates the surface. This means that highly precise models are hard to create. However, we are not

interested in high accuracy. We are interested in producing surface models that roughly maintain the shape of the original object.
User error can also cause erroneous or imperfect models. If the user does not map straight lines very straight, unpleasant results can occur as seen in figures A9 and A10 in Appendix A. If the user accidentally drops or releases the tracker, the object will map the path of the falling tracker, thus making the previous mapping useless. This could easily be solved by being able to erase previously mapped parts of the model.

6.3. Future work

As stated earlier, this application could benefit from various improvements. Most of these improvements involve making the application more user-friendly. Such improvements include the incorporation of a reference object to highlight the position of the tracker in space as the user creates the object. Deleting parts of the generated model could also prove to be useful. Using two tracker would ease the restriction of the modelled object being forced into a static position.
Other improvements can involve sending the data from the modelled object directly into the ShapeData tables. This would considerably reduce the latency problems.

Future work could involve using the available data to create a closed solid model, with emphasis on using a simple and efficient algorithm.

One of the main problems with the application is the rapid creation of large number of polygons. It would be interesting to investigate the possibility of incorporating some method to reduce the complexity of the object. Such

algorithms are widely available [Hoppe *et al* 1993], [Turk 1992], [Schroeder *et al* 1992].

A further extension would be to incorporate spline surfaces into the models. This produces very pleasing results as can be seen in [Loop 1994].

6.4. Conclusion

As a result of this project, RhoVeR has another application that will help in the testing of this VR system. This application provides two tools to digitise real life objects in real time. This process is achieved by tracing the surface of the object with the Polhemus InsideTrak device. The result is a surface model of the mapped object.

References

[Ammeraal 1986] Ammeraal L. (1986) *Programming Principles in Computer Graphics*. John Wiley & Sons.

[Bajaj *et al* 1995] Bajaj C., Bernardini F. and Xu G.(1995). *Automatic Reconstruction of Surfaces and Scalar Fields from3D Scans*. Computer Graphics, (Annual Conference Series, 1995), pp 109-117.

[Bangay *et al* 1996a] Bangay S., Casanueva L., Gain J., Watkins G., and Watkins K. (1996) *RhoVeR: User Documentation*. Rhodes University. http://cs.ru.ac.za/vrsig/

[Bangay *et al* 1996b] Bangay S., Gain J., Watkins G., and Watkins K. (1996) *RhoVeR: Building the Second Generation of Parallel/Distributed Virtual Reality Systems*. First Eurographics Workshop on Parallel Graphics & Visualisation, Bristol(UK), 26-27 September 1996. http://cs.ru.ac.za/vrsig/techdocs.html

[Clark 1976] Clark J. (1976) *Designing Surfaces in 3D.* Communications of the ACM August 1976, 19(8), pp 454-460.

[Foley 1987] Foley J. (1987) *Interfaces for Advanced Computing*. Scientific American, 257(4), October 1987, pp 127-135.

[Foley *et al* 1993] Foley J., Van Dam A., Freiner S. and Hughes J.(1993) *Computer Graphics: Principles and Practice* [2nd Ed, 5th Printing]. Addison-Wesley Publishing Company.

[Gain 1993] Gain J. (1993) *A 3D Sculpt Tool*. B.Sc. (Honours) Thesis, Rhodes University.

[Gain 1996] Gain J. (1996) *Virtual Sculpting: An Investigation of Directly Manipulated Free-Form Deformation in a Virtual Environment*, MSc Thesis, Rhodes University. http://cs.ru.ac.za/vrsig/techdocs.html

[Galyean and Hughes 1991] Galyean T. and Hughes J. (1991) *Sculpting: An Interactive Volumetric Modelling Technique*. Computer Graphics (Proceedings of SIGGRAPH'91), 25(4), pp 267-274.

[Hoppe *et al* 1992] Hoppe H., DeRose t., Duchamp T., McDonald J., Stuetzle W. (1992) *Surface Reconstruction from Unorganised Points*. Computer Graphics (Proceedings of SIGGRAPH'92), 26(2), pp 71-78.

[Hoppe *et al* 1993] Hoppe H., DeRose t., Duchamp T., McDonald J., Stuetzle W. (1993) *Mesh Optimisation*. Computer Graphics (Annual Conference Series, 1993), pp 19-26.

[Loop 1994] Loop C. (1994) *Smooth Spline Surfaces over Irregular Meshes*. Computer Graphics (Annual Conference Series, 1994), pp 303-310.

[McGregor and Watt 1986] McGregor J. and Watt A. (1986) *The Art of Graphics for the IBM PC*. Addison-Wesley Publishing Company.

[Miller *et al* 1991] Miller J., Breen D., Lorensen W., O'Bara R., Wozny M. (1991) *Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data*. Computer Graphics (Proceedings of SIGGRAPH'91), 25(4), pp 217-226.

[Payne and Toga 1994] Payne B. and Toga A. (1994) *Surface Reconstruction by Multiaxial Triangulation*. IEEE Computer Graphics, November 1994, pp 28-35

[Polhemus 3Space 1993] Polhemus Incorporated (1993) *Polhemus 3Space InsideTrak User's Manual*. Polhemus Incorporated.

[Polhemus 3Space 1996] Polhemus Incorporated (1996) *Information about the Polhemus 3Space InsideTrak system*. http://www.polhemus.com

[Preparata and Shamos 1988] Preparata F., Shamos M. (1988) *Computational Geometry: An Introduction* [2nd Printing]. Springer-Verlag.

[Roberts 1966] Roberts L. (1966) *The Lincoln Wand. MIT L incoln Laboratories Report*. Lexington Massachuset, 1966.

[Sachs *et al* 1991] Sachs E., Roberts A. and Stoops D. (1991) *3-Draw:a Tool for Designing 3D Shapes*. IEEE Computer Graphics and Applications, 11(6), pp 18-26.

[Schroeder *et al* 1992] Schroeder W., Zarge J., and Lorensen W. (1992) *Decimation of Triangle Meshes*. Computer Graphics (Proceedings of SIGGRAPH'92), 26(2), pp 65-70.

[Sheng and Meier 1995] Sheng X. and Meier I.(1995) *Generating Topological Structures for Surface Models*. IEEE Computer Graphics and Applications, November 1995, pp 35-41.

[Thomas 1984] Thomas S. (1984) *Modelling Volumes Bounded by B-Splines Surfaces*, Ph.D. Thesis, Technical Report UUCS-84-009, Department of Computer Science, University of Utah, Salt Lake City, UT, June 1984.

[Turk 1992] Turk G. (1992) *Re-Tiling Polygonal Surfaces.* Computer Graphics (Proceedings of SIGGRAPH'92), 26(2), pp 55-63.

[Watkins 1994] Watkins K. (1994) *A Virtual Modelling Environment*, Honours Thesis, Computer Science Department, Rhodes University. http://cs.ru.ac.za/vrsig/techdocs.html

[Wyvill *et al* 1986] Wyvill G., McPheeters C. and Wyvill B. (1986) *Data Structures of Soft Objects*. The Visual Computer, 2(4), pp 227-234.

Bibliography

[Ammeraal 1986] Ammeraal L. (1986) *Programming Principles in Computer Graphics*. John Wiley & Sons.

[Bajaj *et al* 1995] Bajaj C., Bernardini F. and Xu G.(1995). *Automatic Reconstruction of Surfaces and Scalar Fields from3D Scans*. Computer Graphics, (Annual Conference Series, 1995), pp 109-117.

[Banchoff and Wermer 1983] Banchoff T., and Wermer J. (1983) *Linear Algebra through Geometry*. Springer-Verlag.

[Bangay 1994a] Bangay S. (1994) *Creating Virtual Reality Applications on a Parallel Architecture*. Unpublished paper - extract from MSc thesis . Rhodes University. http://cs.ru.ac.za/vrsig/techdocs.html

[Bangay 1994b] Bangay S. (1994) *A Comparison of Virtual Reality platforms. Unpublished paper - extract from research toward PhD currently underway* .

Rhodes University.
http://cs.ru.ac.za/vrsig/techdocs.html

[Bangay *et al* 1996a] Bangay S., Casanueva L., Gain J., Watkins G., and Watkins K. (1996) *RhoVeR: User Documentation*. Rhodes University. http://cs.ru.ac.za/vrsig/

[Bangay *et al* 1996b] Bangay S., Gain J., Watkins G., and Watkins K. (1996) *RhoVeR: Building the Second Generation of Parallel/Distributed Virtual Reality Systems*. First Eurographics Workshop on Parallel Graphics & Visualisation, Bristol(UK), 26-27 September 1996. http://cs.ru.ac.za/vrsig/techdocs.html

[Clark 1976] Clark J. (1976) *Designing Surfaces in 3D.* Communications of the ACM August 1976, 19(8), pp 454-460.

[Foley 1987] Foley J. (1987) *Interfaces for Advanced Computing*. Scientific American, 257(4), October 1987, pp 127-135.

[Foley *et al* 1993] Foley J., Van Dam A., Freiner S. and Hughes J.(1993) *Computer Graphics: Principles and*

Practice [2nd Ed, 5th Printing]. Addison-Wesley Publishing Company.

[Gain 1993] Gain J. (1993) *A 3D Sculpt Tool*. B.Sc. (Honours) Thesis, Rhodes University.

[Gain 1996] Gain J. (1996) *Virtual Sculpting: An Investigation of Directly Manipulated Free-Form Deformation in a Virtual Environment*, MSc Thesis, Rhodes University. http://cs.ru.ac.za/vrsig/techdocs.html

[Galyean and Hughes 1991] Galyean T. and Hughes J. (1991) *Sculpting: An Interactive Volumetric Modelling Technique*. Computer Graphics (Proceedings of SIGGRAPH'91), 25(4), pp 267-274.

[Hearn and Baker 1986] Hearn D, and Baker P. (1986) *Computer Graphics*. Prentice-Hall International.

[Hoppe *et al* 1992] Hoppe H., DeRose t., Duchamp T., McDonald J., Stuetzle W. (1992) *Surface Reconstruction from Unorganised Points*. Computer Graphics (Proceedings of SIGGRAPH'92), 26(2), pp 71-78.

[Hoppe *et al* 1993] Hoppe H., DeRose t., Duchamp T., McDonald J., Stuetzle W. (1993) *Mesh Optimisation*. Computer Graphics (Annual Conference Series, 1993), pp 19-26.

[Hoppe *et al* 1994] Hoppe H., DeRose t., Duchamp T., Halstead, M., Jin H., McDonald J., Schweitzer J., Stuetzle W. (1994) *Piecewise Smooth Surface Reconstruction*. Computer Graphics (Annual Conference Series, 1994), pp 295-302.

[Loop 1994] Loop C. (1994) *Smooth Spline Surfaces over Irregular Meshes*. Computer Graphics (Annual Conference Series, 1994), pp 303-310.

[McGregor and Watt 1986] McGregor J. and Watt A. (1986) *The Art of Graphics for the IBM PC*. Addison-Wesley Publishing Company.

[Miller *et al* 1991] Miller J., Breen D., Lorensen W., O'Bara R., Wozny M. (1991) *Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data*. Computer Graphics (Proceedings of SIGGRAPH'91), 25(4), pp 217-226.

[Parslow *et al* 1975] Parslow R., Prowse R., Elliot Green R. (1975) *Computer Graphics: Techniques and Applications*. Plenum/Rosseta Publishing Company.

[Payne and Toga 1994] Payne B. and Toga A. (1994) *Surface Reconstruction by Multiaxial Triangulation*. IEEE Computer Graphics, November 1994, pp 28-35

[Polhemus 3Space 1993] Polhemus Incorporated (1993) *Polhemus 3Space InsideTrak User's Manual*. Polhemus Incorporated.

[Polhemus 3Space 1996] Polhemus Incorporated (1996) *Information about the Polhemus 3Space InsideTrak system*. http://www.polhemus.com

[Preparata and Shamos 1988] Preparata F., Shamos M. (1988) *Computational Geometry: An Introduction* [2nd Printing]. Springer-Verlag.

[Roberts 1966] Roberts L. (1966) *The Lincoln Wand. MIT L incoln Laboratories Report*. Lexington Massachuset, 1966.

[Sachs *et al* 1991] Sachs E., Roberts A. and Stoops D. (1991) *3-Draw:a Tool for Designing 3D Shapes*. IEEE Computer Graphics and Applications, 11(6), pp 18-26.

[Schroeder *et al* 1992] Schroeder W., Zarge J., and Lorensen W. (1992) *Decimation of Triangle Meshes.* Computer Graphics (Proceedings of SIGGRAPH'92), 26(2), pp 65-70.

[Sheng and Meier 1995] Sheng X. and Meier I.(1995) *Generating Topological Structures for Surface Models*. IEEE Computer Graphics and Applications, November 1995, pp 35-41.

[Thomas 1984] Thomas S. (1984) *Modelling Volumes Bounded by B-Splines Surfaces*, Ph.D. Thesis, Technical Report UUCS-84-009, Department of Computer Science, University of Utah, Salt Lake City, UT, June 1984.

[Turk 1992] Turk G. (1992) *Re-Tiling Polygonal Surfaces.* Computer Graphics (Proceedings of SIGGRAPH'92), 26(2), pp 55-63.

[Watt 1990] Watt A. (1990) *Fundamentals of Three-Dimensional Computer Graphics*. Addison-Wesley Publishing Company

[Watt and Watt 1993] Watt A., Watt M. (1993) *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley Publishing Company.

[Watkins 1994] Watkins K. (1994) *A Virtual Modelling Environment*, Honours Thesis, Computer Science Department, Rhodes University. http://cs.ru.ac.za/vrsig/techdocs.html

[Wyvill *et al* 1986] Wyvill G., McPheeters C. and Wyvill B. (1986) *Data Structures of Soft Objects*. The Visual Computer, 2(4), pp 227-234.

[Yaqub and Moore 1980] Yaqub A., and Moore H. (1980) *Elementary Linear Algebra with Applications*. Addison-Wesley Publishing Company