A Virtual Modelling Environment

by Kevan Watkins

Computer Science Honours 1994

Rhodes University

A Virtual Modelling Environment

Submitted in partial fulfilment

of the requirements of the degree of

Bachelor of Science (honours)

of Rhodes University

by

Kevan Watkins

November 1994

Abstract

This thesis considers the problem of developing solid modelling systems that are more efficient and easier to use.

A virtual reality (VR) approach is taken to make the use of modelling tools more intuitive and interactive.

This work is motivated by the increase in demand for complex solid objects, caused mainly by the rapid increase in the number of VR applications. This demand can only be met by making solid modelling systems easy to use, and therefore usable by more people.

Table of Contents

Chapter 1
Introduction

1.1. Project Goal

Research on solid modelling has lead to the development of many powerful tools for creating and manipulating solid objects. The lack of an intuitive and interactive interface for using these tools has, however, impeded the effectiveness of solid modellers.

Since the increase in popularity of virtual reality in 1989, much research has been done in developing highly intuitive and interactive user interfaces to computer programs. This has lead to significant advances in both input and output devices as well as techniques for using these devices.

The intention of this project is to investigate the combination of a virtual reality interface and solid modelling tools to form an efficient and easy to use solid modelling system.

1.2. Solid Modelling

Solid modelling is by no means a new field in computer graphics. Much research has been done, and many techniques have been developed for creating and modifying solid objects on computer. There have until recently, however, been several hardware restrictions which have caused modelling systems to be less effective than they would otherwise have been.

The main restriction was with the input devices. Modellers were forced to use a two dimensional (2D) input device such as a mouse for the building of three dimensional (3D) objects. This restriction caused modelling systems to be counter-intuitive. They also typically required the use of extensive menu systems, which made the systems non-interactive [Req 92].

3D objects cannot be uniquely and unambiguously represented on a two dimensional screen. For this reason, complex viewing techniques, such as the tri-view system used in engineering, are often used to convey 3D information to the user. This, together with the lack of intuitiveness and interaction, has caused most modelling systems to be usable only by a small number of experts.

With the recent advent of VR, and in particular 3D input and output devices, it has become possible to use 3D solid modelling tools and techniques in a direct, interactive way. Research on developing such systems is, however, still in its infancy. [Rossignac 91]

The rapid increase in the number of virtual reality applications has caused an increase in demand for complex 3D models.[Sproull 90] This, in turn, has increased the need for more effective, easy to use, solid modelling systems.

1.3. Virtual Reality

Human beings are accustomed to viewing and interacting with three dimensional worlds [Balaguer].
We recognize three dimensional shapes almost instantly. We touch, pick up, move, or in some way interact with three dimensional objects almost continuously throughout our lives.

With Virtual Reality, we attempt to make human vision and interaction with a computer generated world similar to that of the real (physical) world. In this way we enable the user to use his/her experience in interacting with the real world instead of requiring him/her to learn to perceive and interact with a vastly different world.

The most important characteristic of VR is immersion. As discussed in [Balaguer] and [Bricken 90], when the user is surrounded by a computer generated world, his perception of the images is dramatically altered. His feelings change from that of looking at a screen or picture to that of being in a place or inhabiting an environment.

One step in immersing the user in the computer generated world can be achieved by using a stereoscopic display. This gives the user better spatial perception of the world and enables him to view an object as he would in the real world.

The most effective way known to date of providing the user with the sense of immersion and presence is with the use of an advanced Head Mounted Display (HMD). With a stereoscopic display and head tracking, the user can alter his orientation in the computer generated world in a very intuitive way. He can simply turn or move his head, just as he would in the real world.

The most powerful tool we humans have for interacting with the physical world are our hands. An input device which allows the user to use his hands in a computer generated environment in a way similar to how he uses them in reality is therefore sought. One input device which makes this possible is the data glove. This device measures the position and orientation of the user's hand as well as the flex of each finger. This information can be used to generate a virtual copy of the hand within the computer generated world. The virtual hand can then be used to interact with the rest of the virtual world.

1.4. VR devices available

Input and output devices are important for any virtual reality system. They provide the user with the only link he has with the computer generated world.

The devices available for this project are discussed below.

Figure 1 : The Power Glove

1.4.1. Power Glove

The Power Glove (see Figure 1) is a low end VR glove. It uses ultrasound to detect the position and orientation of the users hand. In addition, it uses strain gauges to detect the flex of the fingers.

The limitations of the Power Glove are :

- The orientation of the hand is only measured around one axis.
- The finger flex is measured only in the thumb and first three fingers. In addition, no measurement is taken of the angle separating the fingers.
- The precision of the position tracking is far less than that of high end gloves.

The Power Glove does, however, detect enough to enable intuitive use of the users hand. Several common hand gestures are easily detectable. Precision is sufficiently high for applications such as aesthetic solid modelling, where accuracy is not of vital importance.

1.4.2. Head Mounted Display

A simple head mounted display (see Figure 2) has been constructed at the Rhodes University Computer Science Department. The HMD consists of two colour LCD TV screens mounted on a helmet. A VGA to PAL converter is connected to each TV. This converter allows VGA output from a computer to be displayed on a TV screen.

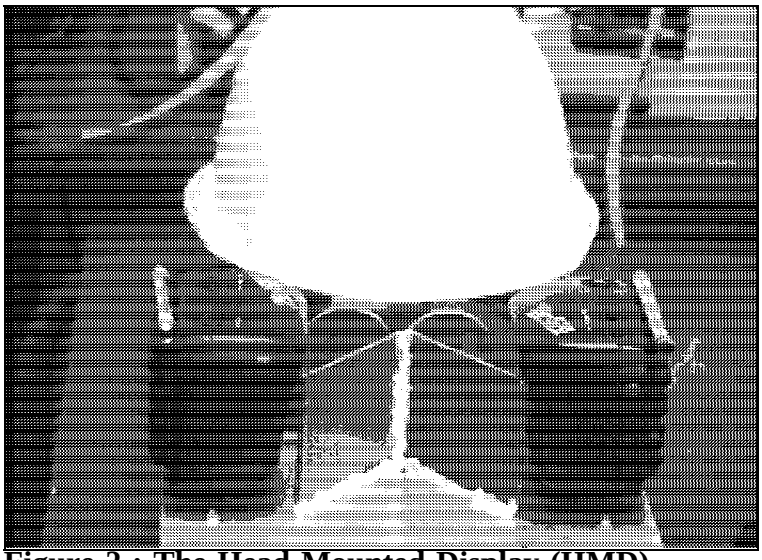

Figure 2 : The Head Mounted Display (HMD)

The greatest disadvantages of this HMD over those used in most high end VR systems is the lack of head tracking. Without head tracking, the user cannot simply turn his head to look in another direction in virtual space. Although the same result can be achieved with an additional device such as a joystick, the sense of presence is diminished. Another shortfall of this HMD is the low resolution of the TV screens (about 200x160), which decreases the realism of the images displayed.

Although this HMD is rather crude, it does enable colour stereoscopic vision of a computer generated world. This alone can remove the feeling of looking at a display and create one of being surrounded by a world. It also removes the ambiguity inherent in a 2D view of a 3D object.

1.5. Project structure

VR-386 is used as a base for the implementation of this project. It is a polygon rendering library for the 386 and 486 computers with a VGA display. It supports the Power Glove, joystick and other devices. In addition, object support such as rotation, loading and saving is provided. VR-386 was written by Dave Stamp.

This project consists of three main steps.

- Developing a library of solid modelling tools which operate on objects stored and rendered by VR-386.
- Extending VR-386 to provide support for the HMD mentioned in section 1.4.2.
- Developing a Virtual Modelling Environment. This involves developing techniques for using the modelling tools with the input devices.

Solid modelling tools are discussed in chapters 2 and 3. The suitability of various tools to use in a virtual reality environment are evaluated. Several tools are then selected for implementation in the modelling system. Finally, the selected tools are considered further. Their implementation and optimization is discussed.

Chapter 4 deals with the software support for the head mounted display. Various optimization techniques are discussed and their performance evaluated.

The Virtual Modelling Environment as a whole is discussed in chapter 5.

Chapter 2

Tools for Solid Modelling

In creating solid objects, people use various tools and techniques. To create a clay sculpture, for example, one might use his hands, a lathe, and a rasping tool. A carpenter might use a saw, various chisels, and sandpaper.

We also need various tools or modelling techniques when working in a Virtual Modelling Environment.

In this chapter, the properties that make a tool useful in a virtual modelling environment are identified. Several modelling tools are then analysed and compared. Finally a selection of tools is made. Motivation for these selections is given and their implementation is discussed.

2.1. Desirable properties

When a sculptor makes a sculpture he does so in two main stages. First he forms a rough model of the subject. This model shows the basic shape or form of the subject, but typically has very little detail. He then completes the sculpture by adding the necessary detail or texture to the rough model.

The sculptor also typically uses different tools and different techniques for the different stages. A clay sculptor for instance, often uses his bare hands for the initial moulding and fine scraping tools for the last stage.

A similar approach is adopted by carpenters, stone workers and many other solid object creators.

People working in a Virtual Modelling Environment are also likely to want to adopt this 'make and refine' approach to solid modelling.

For this reason, each tool used in a Virtual Modelling Environment should be useful in either the initial, rapid construction, stage or the second, refinement, stage. In addition, there should be a sufficient set of tools for use in each stage.

For a tool to be useful in the first stage, it must substantially modify or add to the overall shape of an object.
To be useful for the second stage, a tool should alter the finish or texture of an object or enable the fine adjustment of parts of the object.

In addition to satisfying one of the above conditions, a tool should have the following six properties in order to be useful in practice.

- **Intuitive**. The tool must be able to be used in a simple and natural way. This typically requires that the tool use few or no attributes that need to be explicitly set by the user. The number of attributes that have to be explicitly set may depend on the input devices used.

- **Interactive**. A tool is usually much more effective if it can be applied continuously and interactively by the user. Many tools can be made fairly interactive by the use of handles and sliders. The best interaction is, however, achieved when the tool allows the user to interact directly with the object.

- **Predictable**. The results of using the tool should be consistent and predictable by the user.

- **Computationally Inexpensive**. The complexity of any computation done by the tool should be low enough to enable real-time interaction. Ideally, any delay caused by the computation should be well under one tenth of a second.

  The time taken to perform a specific computation is determined by the speed of the computer system used. The tolerated computational complexity of a tool is, therefore, also determined by the speed of the computer system.

- **General**. The tool should be general enough to be useful in the making of a wide range of objects.

- **Have a Widespread Effect**. A tool for use in either stage of object creation is most efficient if it has a big or widespread effect on the object. For a tool used in the first stage, this means that it should substantially alter or add to the shape of the object. For a tool used in the final stage to have a widespread effect, it should alter the texture or finish of a large area of the object's surface.

2.2. Analysis of Several Tools

Although most of the tools listed below are not specific to one representation, they can all be implemented using a polyhedral representation of solids. This representation is discussed in detail in chapter 3.

In considering the computational complexity of a given tool, only the implementation with polygonal objects was taken into account. The reason for this is that this implementation is most likely to be the least complex. Rendering of polyhedral objects is also much faster than rendering of most other representations of solids.

2.2.1. Explicit Definition

This is a technique used by many existing polygonal solid modellers. First the vertices are explicitly defined or placed. Polygons, which make up the final object, are then constructed using these vertices. Explicit definition is not suitable for use in a virtual environment as it is not intuitive or interactive. Although it is possible to make any object with this tool, it is only practical for constructing very simply ones because the process is very slow.

2.2.2. Extrusion Tool

An example of an extrusion tool is given in [Butterworth 92].
First the user traces out the perimeter of an area. The area is then 'grabbed' and moved. The result is the volume traced out by the area. This tool enables the user to create complex objects rapidly if an input device is used that allows arbitrary rotation of the area during extrusion.

Once the initial step of drawing the perimeter is completed, the tool becomes very intuitive and interactive, allowing fairly rapid construction of objects.

2.2.3. Regular Shape Creation

This involves the creation of regular polyhedra such as globes, cones, cylinders, boxes and tori.

Although this is not a very powerful tool in itself, it can be useful in conjunction with other tools such as the Boolean set operations described below.

2.2.4. Boolean Set Operations

Here we refer to the union, intersection and difference of two volumetric objects.

Although these tools are fairly commonly used, the two objects are usually explicitly and non-intuitively placed. With the use of an interactive virtual environment with 3D input and output devices, this tool can be much more effective.

By continually applying one of the Boolean operations, we can build very effective modelling tools. Two such tools are listed below.

2.2.5. Subtractive Tool

The subtractive tool, as mentioned in [Galyean 91], involves the use of an 'eraser'. The set difference (New Object = Old Object - eraser) is used continuously to remove parts of an object that are not wanted.

The subtractive tool is very intuitive and interactive. The main problem with this tool is that the resulting objects are very rough, requiring the use of powerful finishing tools such as smoothing tools to obtain a good result. The computational complexity of the subtractive tool is also fairly high because it uses a Boolean set operation.

2.2.6. Additive Tool

This tool, also mentioned in [Galyean 91], consists of an object that leaves volume everywhere it is moved. An analogy is that of a foam spray which leaves foam suspended wherever it moves.

The additive tool continually applies the union of two volumes to obtain its functionality (New Object = Old Object U additive object). This tool, like the subtractive tool, is very intuitive and

interactive, allowing the rapid construction of objects, but it has the same setbacks as the subtractive tool.

2.2.7. Pulling/Pushing Tools (single point)

This tool, discussed in [Bill 94] and others, involves the pulling or pushing of a single point on the surface of an object. Various decaying functions can be applied to cause different effects on the surface surrounding the point.

The pulling/pushing tool allows an object to be directly manipulated by the user. This enables the user to make the object take form rapidly.

One problem with this tool is that the results are not as predictable as they are with most other tools. The computation required by the pulling/pushing tool can also be fairly high.

2.2.8. Pulling/Pushing Tools (multiple point)

This tool is an extension of the single point tool mentioned above. It allows several points on an object to be simultaneously pulled or pushed in different directions.
With the use of this tool and an advanced glove input device, we can enable the full-hand manipulation of an object. If implemented correctly, it can be the most intuitive and interactive of the tools mentioned in this chapter, allowing the rapid construction of complex objects.

One disadvantage of this tool is that it requires really advanced, and therefore expensive, input devices to obtain maximum intuitiveness and interaction. Another disadvantage is that it uses very heavy computation. It therefore requires an expensive computer system to obtain the speed required for immersive interaction.

2.2.9. Direct Point Moving

This is another tool mentioned in [Butterworth 92]. It involves the selection and moving (translating and rotating) of a set of vertices. This tool is best applied to objects constructed purely of triangles. The reason for this is that if a polygon has more than three vertices, the translation of one or more of the vertices can make the set non-coplanar, thus making the polygon invalid.

Direct point moving is fairly interactive once the points have been selected and are being moved. It is suited more to the final stage of object creation than the initial stage because it allows fine adjustment of existing objects and cannot rapidly form or alter the shape of complex objects.

2.2.10. Smoothing Tool

A smoothing tool is discussed in [Bill 94] and [Galyean 91]. This tool is useful in the final stage of object creation, and is analogous to sandpaper used in woodwork. It could be implemented by applying a smoothing function to the surface of either the entire object, or just that part of the object which is overlapping with a virtual sanding block.

A comparison of the tools discussed above is given in table 1.

Table 1 : A comparison of several solid modelling tools.

Tools	Properties					
	Intuitive	Interactive	Predictable	Inexpensive	General	Widespread
Explicit Definition	1	1	2	5	2	1
Extrusion	4	4	5	3	3	6
Regular Shapes	4	4	5	3	3	5
Boolean Operations	4	3	5	2	3	2
Additive	5	5	5	2	5	5
Subtractive	5	5	4	2	5	4
Pull/Push (sngl pnt)	5	5	4	2	4	4
Pull/Push (mtpl pnt)	6	6	4	1	5	5
Direct Point Moving	3	3	5	5	3	3
Smoothing	5	4	5	2	4	5

KEY

1 not
2 very slightly
3 slightly
4 moderately
5 very
6 exceptionally

2.3. Selection

Tools designed for use in the final stage of object creation typically require more accurate input than tools for the first stage. The Power Glove, although it is a useful, intuitive device, does not enable very accurate input. It was therefore decided to focus mainly on the first stage of object creation and leave the final 'touch up' stage as a possible extension.

Even in the first stage, there is a need to limit the number of tools available. The reason for this is that an environment with many vastly different tools and modelling techniques can cause user interaction to be less consistent. This in turn causes the environment to be less intuitive.

We chose to implement the additive and subtractive tools that use the Boolean set operations as a base (see section 2.2), as well as the single Boolean operations. In addition, regular shape creation was implemented to provide basic objects on which the user can apply these tools.

The main reasons for the selection are listed below.

- Relatively low implementation cost. Both the additive and subtractive tools rely purely on Boolean set operations for their functionality. Consequently, their combined implementation costs little more than the implementation of the Boolean set operations.

- The tools are complementary. Often, the use of different tools for the construction of different parts of an object can make the parts look too dissimilar or too obviously constructed with different tools. With the additive and subtractive tools, however, the resulting objects look very similar. They can thus both be used in the construction of a single object.

- Generality. As discussed in 2.2, both the additive and subtractive tools can be used in the construction of a vast range of objects. Together, because they complement each other well, and can therefore both be used in the construction of a single object, they cover an even wider range of application.

- Relatively low computational complexity. Because the system is to be implemented on a relatively low powered computer system, it is important that each tool has a fairly low complexity. On the other hand, tools with low complexities tend to be much less intuitive

and less powerful than those with high complexities. The additive and subtractive tools provide a reasonable compromise between these two ideals.

Chapter 3

Boolean set operations

3.1. Introduction and Definitions

If we define an ***object*** as a set of points in 3-space e.g.

$$A,B = \{x_1,x_2,\ldots,x_n : x_i \in \mathbb{R}^3;\ i=1,2,\ldots,n\}$$

then we can create new ***objects*** by taking the Boolean set operations of A and B i.e.

$$A \cup B = \{x: x \in A \vee x \in B\}$$
$$A \cap B = \{x: x \in A \wedge x \in B\}$$
$$A \setminus B = \{x: x \in A \wedge x \notin B\}$$

An example of two objects and the results of the Boolean set operations on them is given in Figure 3.

The problem with the ordinary Boolean set operations is that applying them on two solid objects does not always result in a solid object. For example, the Boolean intersection of (a), (b) and (c) of Figure 4 result in a plane, line and point respectively.

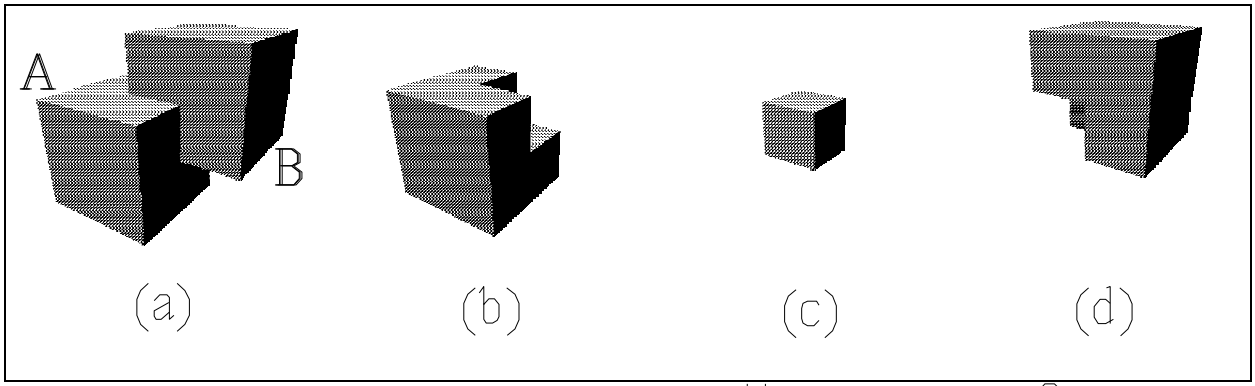

Figure 3 : Boolean operations on solid objects. (a)=A∪B, (b)=A\B, (c)=A∩B, (d)=B\A

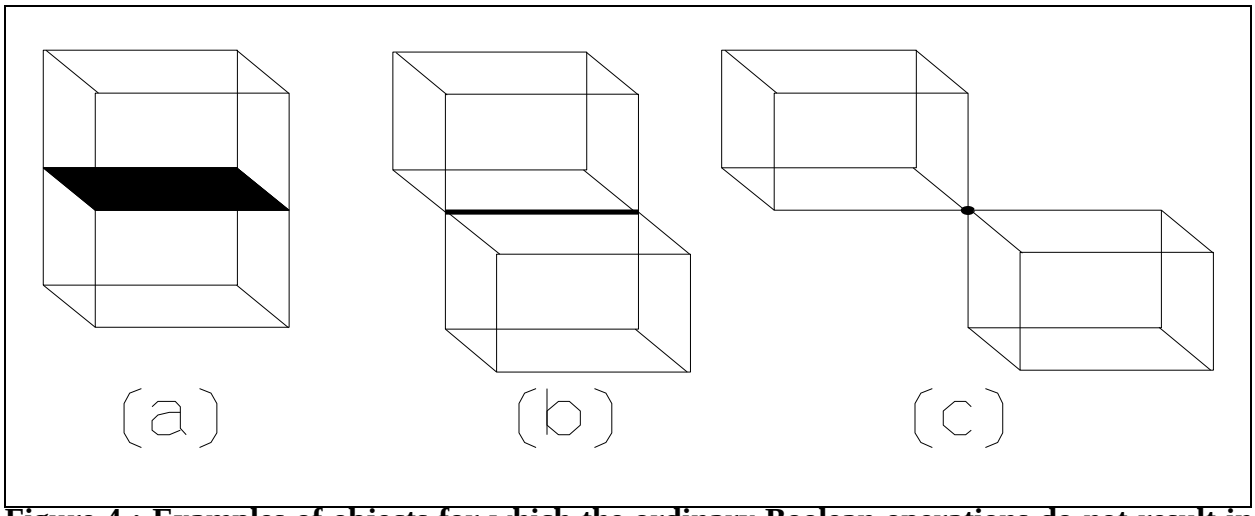

Figure 4 : Examples of objects for which the ordinary Boolean operations do not result in solid objects. The intersection of (a), (b) and (c) form a plane, a line and a point respectively.

We therefore use the regularized Boolean set operations, $\cap^*$, $\cup^*$ and $\setminus^*$. Applying the regularized Boolean set operations on two regular solids always results in a regular solid. A regular solid is one which is the closure of its interior [Foley 91]. Applying the regularized Boolean set operations on the pair of objects in Figure 3 produces the same results as the ordinary Boolean set operations. Applying the regularized Boolean intersection or set difference to any of the object pairs in Figure 4, however, results in the empty object.

So far, we have dealt only with objects defined as sets of points in space. Because regular objects are the closure of their interior, they can be represented by their boundaries. An object is usually represented by dividing its boundary into subsets called faces, delimited by edges. In the case of polyhedral objects, the faces are polygons, and the edges are line segments.

The following theorem is formally proved in [Requicha 78].

> The boundary of A <op> B is a subset of the boundaries of the operands A and B. Where <op> is either the $\cap^*$, $\cup^*$ or $\setminus^*$ operation.
> Generalizing, the boundary of any object composition is included into the union of the primitive boundaries.

The boundary representation of the resultant object can therefore be constructed purely by combining parts of the boundaries of the input objects.

The rest of this chapter deals with algorithms for applying the regularized Boolean set operations to polyhedral objects.

3.2. Classification of algorithms

Algorithms for evaluating Boolean set operations on boundary representations of objects can be divided into three classes [Badouel 88] :

- Edge/Face Each edge/face intersection is evaluated. Truncated polygons are then closed by creating new edges. Each polygon is then classified as being either inside or outside the volume of the other object. In so doing, four sets of polygons are formed : AinB, AoutB, BinA and BoutA.

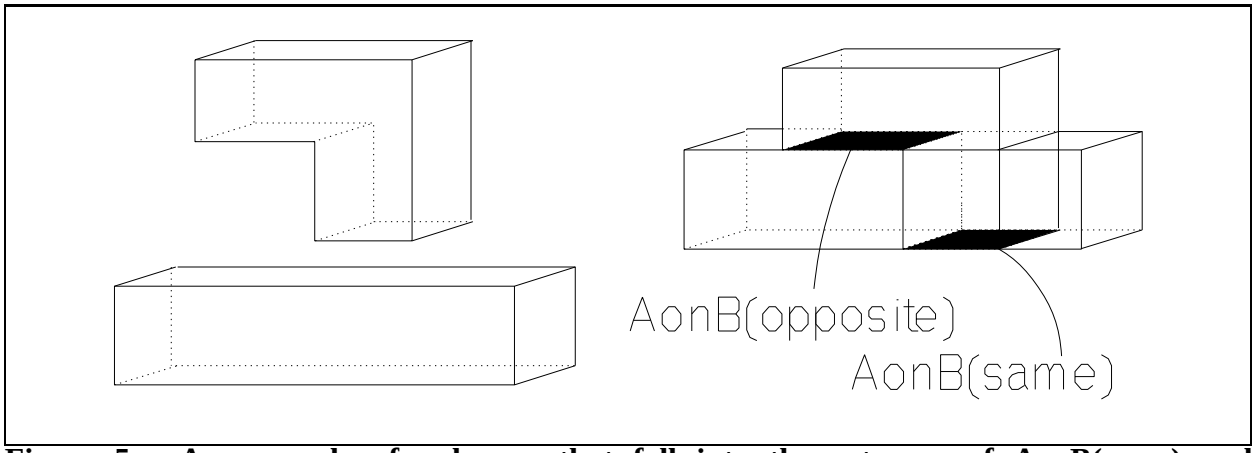

Figure 5 : An example of polygons that fall into the category of AonB(same) and AonB(oposite).

The Boolean set operations are then formed as follows :

$A \cup^* B$ = AoutB, BoutA

$A \setminus^* B$ = AoutB, BinA

$A \cap^* B$ = AinB, BinA

- Edge/Solid Each edge is intersected with the other object (solid). A divide-and-conquer strategy is then used to determine the neighbourhood of each edge (i.e. if it is inside, on or outside the volume of the resultant object). Polygons are then constructed from those edges that fall on the boundary of the desired volume.

- Face/Face Each face/face intersection is resolved. The resulting polygons are then divided into six sets. The first four sets are the same as those used in the Edge/Face class. The other two sets are AonB(same) and AonB(opposite). These two sets contain those polygons resulting from the intersection of coplanar polygons. AonB(same) contains the intersection of polygon pairs with solid on the same side. AonB(opposite) contains the intersection of polygon pairs with solid on opposite sides. An example of this classification is shown in Figure 5.

  The Boolean set operations are formed as follows :

  $A \cup^* B$ = AoutB, BoutA, AonB(same)

  $A \setminus^* B$ = AoutB, BinA, AonB(opposite)

  $A \cap^* B$ = AinB, BinA, AonB(same)

Edge/Face algorithms do the main construction of the resultant object fairly efficiently. This construction does, however, not ensure closure of the object. By adding a post construction stage, where holes in the boundary are identified and closed, closure can usually be obtained. This, however, detracts from the efficiency of the algorithm. There are also several cases for which closure can still not be obtained.

The Edge/Solid class of algorithms guarantees valid resultant objects. The neighbourhood function is, however, expensive and can have a large effect on the efficiency of the algorithm as a whole.

The algorithms of the Face/Face class are well suited to application in a virtual modelling environment. They guarantee valid resulting objects, and are more efficient than Edge/Solid algorithms.

In the next section, Face/Face algorithms are considered further. The main factors influencing the efficiency of a Face/Face algorithm are discussed. An efficient Face/Face algorithm is presented in section 3.4.

3.3. Optimization of Face/Face algorithms

3.3.1. Constraints on the objects

One way of reducing the time taken to perform the Boolean operations on objects is to place additional constraints on the objects. These constraints should not limit the range of objects that can be represented. Objects resulting from the Boolean operations should also adhere to the constraints so that they can be used as input objects to subsequent Boolean operations.

One constraint commonly placed on objects is that they consist purely of **convex** polygons. Although algorithms do exist for finding the Boolean set operations on objects without this constraint (such as the algorithm presented in [Putnam 86]), they are much less efficient.

Because any non-convex polygon can be decomposed into several convex polygons covering the same area, no generality in object representation is lost. In addition, any non-convex polygon formed during the Boolean operations can be decomposed, ensuring that the resulting object also consists purely of convex polygons.

Evaluating the intersection of two convex polygons requires much less computation than the intersection of two non-convex polygons. Intersection of polygons is one of the most computationally expensive parts of the Face/Face algorithm. A reduction in the complexity of the polygon intersections therefore substantially improves the efficiency of the algorithm.

3.3.2. Intersection of polygons

The first stage in all Face/Face algorithms involves the intersection of polygons. Several approaches have been taken to improving the efficiency of this stage.

One approach is to minimize the time taken in testing if two polygons intersect. The most common way to do this is by calculating and storing the bounding box of each polygon. Instead of testing if two polygons intersect, we first test if their bounding boxes overlap. If the boxes do not overlap then the polygons do not intersect. If the boxes do overlap then the polygons may or may not intersect, and have to be tested in the usual way. Testing if bounding boxes overlap requires much less computation than directly testing the polygons. The bounding box of a polygon also typically overlaps with a small percentage of the bounding boxes of other polygons. This approach therefore substantially reduces the overall time taken to perform the intersections.

A second approach is to minimize the number of polygon pairs that are tested.
An example of such an approach is presented in [Badouel 88]. The basic idea is to divide the space surrounding the objects into regions. Polygons are then sorted into the appropriate regions. Each polygon can only intersect with those polygons that share at least one region. In this way, the number of other polygons that each polygon has to be tested against is substantially reduced.

3.3.3. Set membership of polygons

The other main part of Face/Face algorithms is the classification of polygons into the six subsets mentioned in section 3.2.
Polygons that belong in AonB(same) or AonB(opposite) can be detected during the intersection stage, and require very little computation.

Grouping polygons into the other four sets reduces to the problem of separating the polygons of an object that lie inside the other object from those that lie outside. The task of determining if a given polygon lies inside or outside the other object is, in general, an expensive one. The algorithm presented in the next section eliminates the need for testing all individual polygons, and thereby dramatically reduces the computation required.

3.4. An efficient Face/Face algorithm

The following algorithm is based on the one used by IRIT, a freeware solid modelling program written by Gershon Elber. The algorithm is modified to allow all valid inputs, and is slightly optimized. The algorithm requires that the input objects be closed and contain only convex polygons. Each polygon must also be defined in such a way that we can determine which side of the polygon is in the object and which is outside. This is usually achieved by defining the polygons to have a set orientation as viewed from the outside. A normal vector to the polygon can then be constructed that points outside the object.

The algorithm can be divided into six main steps.

- **Compute adjacencies**
- **Intersect polygons**
- **Build open/closed loops**

- **Truncate intersected polygons**
- **Propagate from truncated polygons**
- **Decompose non-convex polygons**

Each of these steps is discussed below.

3.4.1. Compute adjacencies

Two polygons are adjacent if and only if there is an edge common to both of them.

The first step of the algorithm is to find and store all adjacencies. These adjacencies should be stored in such a way that, given a polygon, we can access all other polygons adjacent to it. We must also know which edge is shared with each adjacent polygon.

The brute force method for solving this problem is to test each edge of each polygon against each edge of every other polygon. This algorithm is in $O(n^2)$, where n is the number of edges. By using a hash table, however, the problem can be solved in O(n). With this approach, a hashing function is used to sort all edges into a hash table. The table is then traversed and matching edges are handled. An example of a hashing function is the projection of the midpoint of the edge onto a given vector.

Some adjacencies are broken (deleted) during the second and fourth stages (sections 3.4.2 and 3.4.4) of the algorithm. The remaining adjacencies are used in the fifth stage to propagate from a subset of the resultant objects boundary to its entire boundary.

3.4.2. Intersect polygons

In this stage, all polygon/polygon intersections are evaluated.

For the purpose of this algorithm, two polygons are considered to intersect iff there is a common point in the interior of both polygons. The polygon pairs ***a***, ***b*** and ***c*** of Figure 6 are therefore not considered to intersect.

There are two main cases of polygon/polygon intersections. The first case (shown in part **d** of Figure 6) is where the intersection of the polygons is a line segment. The second case (shown in part **e** of Figure 6) occurs where the polygons are coplanar. In this case, their intersection is a polygon. Both cases are considered below.

3.4.2.1. Coplanar polygon intersection

First we test if the intersection polygon (i.e. the overlapping region) should be included in the resultant object. To do this, we test the normals of the polygons. If they point in the same direction, then there is solid on the same side of both polygons, and the intersection polygon is in AonB(same). Similarly, if the normals point in opposite directions, then the intersecting polygon is in AonB(opposite). If we are performing the union or intersection, and the polygon is in AonB(same), then it is included in the resultant object. Alternatively, if we are performing the set difference, and the polygon is in AonB(opposite), then it is included in the resultant object.

The two original polygons are then truncated so that they do not include the overlapping region.

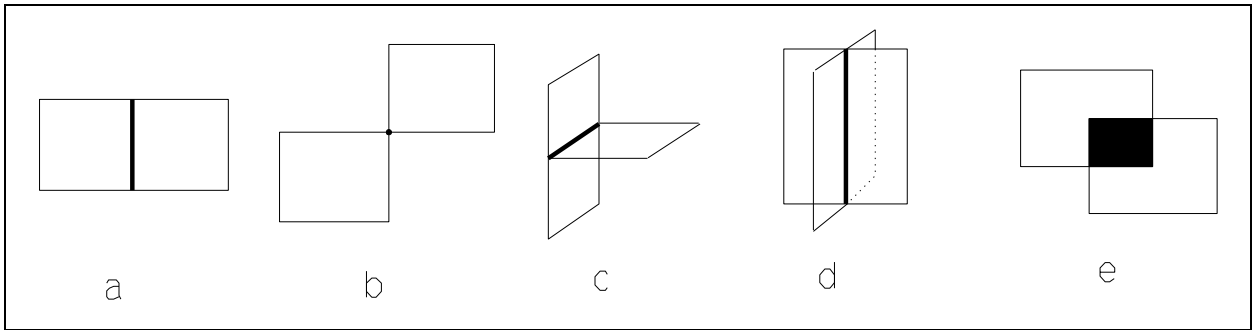

Figure 6 : Intersection of polygons. a, b and c have no common interior points. d has an intersection line segment. The polygons in e are coplanar and their intersection is a polygon.

.4.2.2. non-coplanar polygon intersection

If the two intersecting polygons are not coplanar, then they intersect along a line segment. This line segment is computed and stored with both polygons. Access to the other intersecting polygon should also be provided.

If two polygons do not intersect, but one polygon falls exactly on the edge of the other, then the adjacencies associated with that edge are broken. A pointer, or some access information, to each of the two polygons is stored in a list to be used in the fifth stage (section 3.4.5). If the intersection only contains the edge of one of the polygons, then the edge is stored as an intersection edge with the other polygon.

At the end of this stage, each polygon has a set of intersection line segments, each with access to the intersecting polygon.

These sets are ordered into lists in the next stage. Access to the intersecting polygons is used in stage five (section 3.4.5).

3.4.3. Build open/closed loops

Because the input objects are closed, the lines formed by the intersection of their boundaries are also closed.
Each intersection line may fall within one polygon (as in (a) of Figure 7), or across several polygons (as in (b) of Figure 7). There are, therefore, two types of intersection loops within a polygon :

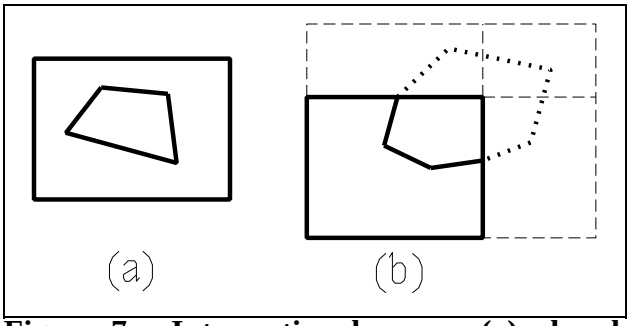

Figure 7 : Intersection loops. (a)=closed, (b)=open.

- closed loop, where the loop has no open end. A closed loop falls completely inside the polygon.
- open loop. In this case, the loop has two open ends. Both ends fall on the boundary of the polygon.

This stage involves forming open and closed loops from the set of line segments formed in the previous stage. This is done by linking segments that have a common endpoint.

3.4.4. Truncate intersecting polygons

In this stage, all intersecting polygons are truncated. The intersection loops are used to discard the parts of each polygon that should not be in the resultant object.

The truncation is done in two main stages. The first stage deals only with the open intersection loops, and the second stage deals with the closed loops.

These two stages are considered in the next two sections.

3.4.4.1. Truncation using open loops

An open loop is internal iff there is a path along the polygon boundary joining its ends, that contains no other loop ends. L1 and L3 in Figure 8 are examples of internal open loops.

An algorithm for truncating a polygon with its open loops is given below.

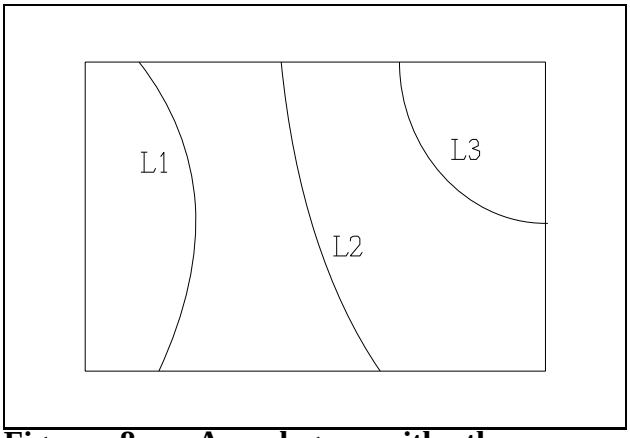

Figure 8 : A polygon with three open intersection loops. L1 and L3 are internal, but L2 is not.

- Find an internal open loop. If no open loop exists, then the truncation is complete.
- Truncate the polygon by replacing the polygon boundary between the endpoints of the loop, with the loop itself. By "the polygon boundary between the endpoints", we refer to that part of the boundary that contains no other open loops.
- If the truncated part of the polygon (the part with no other open loops) belongs in the resultant object, then it is constructed and marked as being in the output object. Determining whether or not the truncated part is in the resultant object is deferred to the end of this section. The construction is done by generating a copy of the open loop, and attaching it to the edges that were removed from the original polygon. If the polygon is not part of the resultant object then it is discarded.
- Repeat the entire process on the rest of the polygon.

In the third step, we determine whether or not the truncated part of the polygon is in the resultant object. To do this, we need to know if the polygon is inside or outside the other input object. This can be determined by testing the first vertex (*v*) of the polygon boundary against the plane through the polygon that caused the last intersection segment in the loop (see Figure 9). First, we calculate the distance of *v* from the plane (in the direction of the normal (*n*) to the plane). If the distance is positive, as in Figure 9, then the polygon is inside the other object, otherwise it is outside.

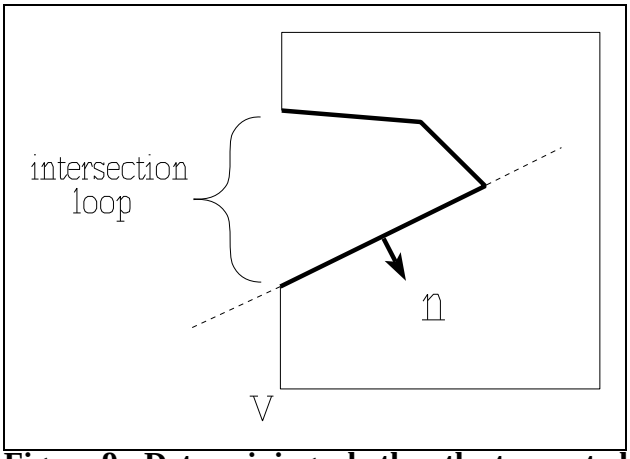

Figure 9 : Determining whether the truncated part of a polygon is in the resultant object.

3.4.4.2. Truncation using closed loops

Each closed loop in a polygon is the result of the intersection of that polygon with the other object. Crossing the edge of a closed loop therefore represents crossing the boundary of the other object. Because no open loops remain, either all the closed loops of a polygon are in the other object, or they are all outside. We can therefore determine if all the loops are either inside or outside the other object by testing a single loop. An algorithm to test if a given closed loop in inside or outside the other object is given below.

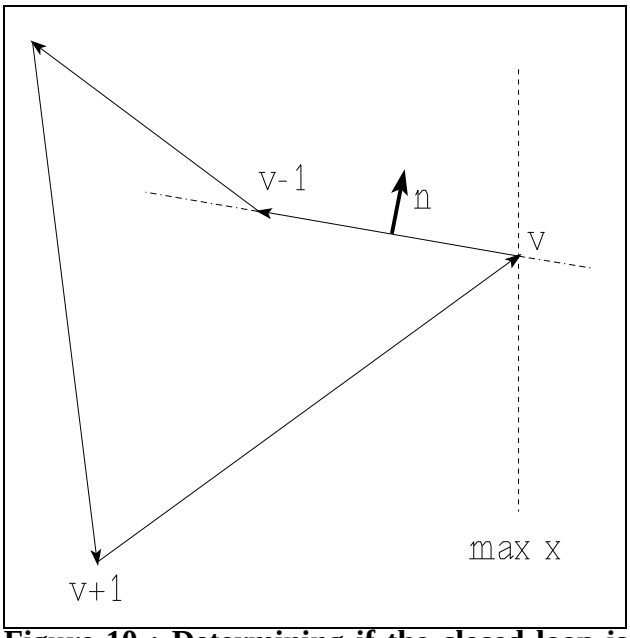

Figure 10 : Determining if the closed loop is inside or outside the other object.

All references made in the algorithm refer to Figure 10

- Transform the polygon and the closed loops so that they lie on the *xy* plane
- Find the vertex (*v*) with the greatest *x* value.
- Consider the next point (*v+1*), and the plane through the polygon that caused the previous intersection segment (<*v-1*,*v*>). Note that the equation of the plane should also be transformed. Calculate the distance (along the direction of the normal (*n*) to the plane) from the plane to the point.
- The closed loop is inside the other object iff the distance calculated in the last step is negative.

If the closed loops have the desired relation with the other object, then a polygon is formed from each closed loop, and the object boundary is discarded.

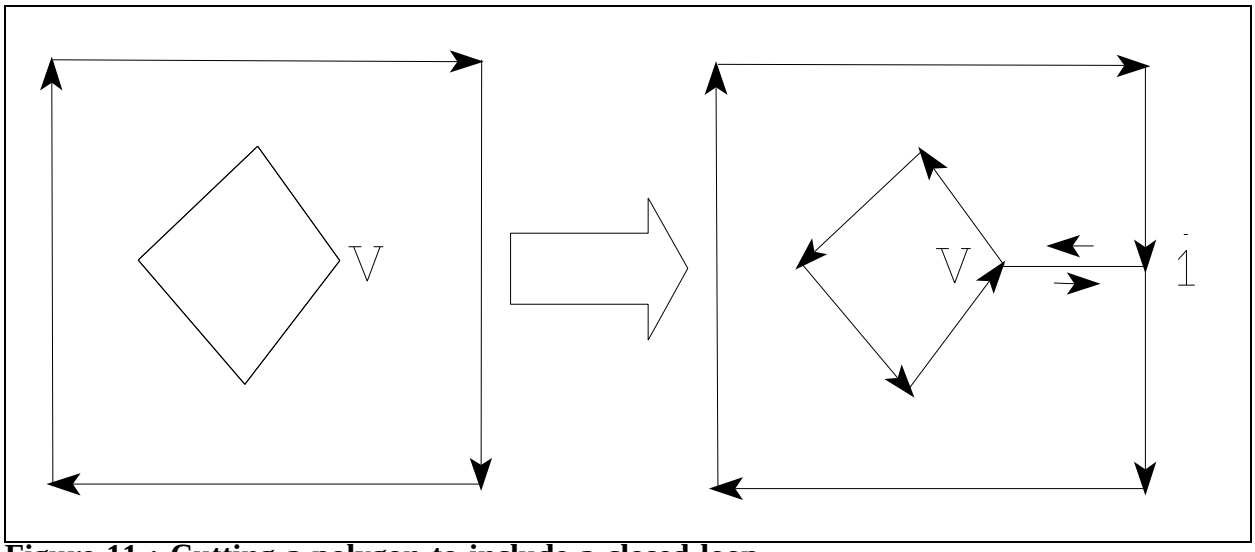

Figure 11 : Cutting a polygon to include a closed loop.

If, however, the closed loops should not be in the resultant object, then they are 'cut' out of the polygon. This procedure is given below and an illustration is given is Figure 11.

- Using the transformed polygon and loops, find the vertex of each loop that has the greatest *x* value.
- Find the loop with the greatest *x* value.
- Make sure that the loop has the opposite orientation to the original polygon boundary, reversing the orientation of the loop if necessary.
- Fire a ray from the vertex (*v*), with the greatest *x* value, in the positive *x* direction, and find its intersection (*i*) with the polygon boundary.
- Break the loop at *v*, and the polygon at *i*. Connect the polygon and the loop by inserting a line segment from *i* to *v*, and another segment from *v* to *i* (see Figure 11).

3.4.5. Propagate from truncated polygons

So far, all work has been done on the intersecting polygons. The result is a set of polygons containing the open edges of AinB, AoutB, BinA and/or BoutA, depending on which Boolean operation is being performed.

In this stage, the inter-polygon adjacencies are used to propagate from these polygons to the other polygons that should be included in the resultant object.

If all intersections of the original objects fall exactly on the edges of one of the object, then there are no intersection polygons to propagate from. In this case, the list built in the second stage (section 3.4.2) is used to find polygons from which we can propagate.

There are two cases where the intersection of polygons falls on an edge. In the first case (illustrated in Figure 12), the edge of one polygon lies in the plane of the other, and inside its boundary. In the other case, the two polygons (from different objects) share an edge or part thereof. Because each polygon has an adjacent polygon in its object (guaranteed by the closure of the objects), there are four polygons sharing the same edge. An example of the second case is illustrated in Figure 13.

In both cases, we must determine if each polygon lies inside or outside the other object. We then keep the polygons with the desired relation to the other object, and propagate from them.

In the first case, only the polygon containing the intersecting edge needs to be tested (polygon B in Figure 12). To test if this polygon is inside or outside the other object, only one vertex needs to be considered. Any vertex other than the two vertices making up the intersection edge, can be used. If the vertex lies on the solid side of the other polygon, then the entire polygon is inside the other object, otherwise it is outside. In Figure 12, vertex *v* can be tested against polygon *A*.

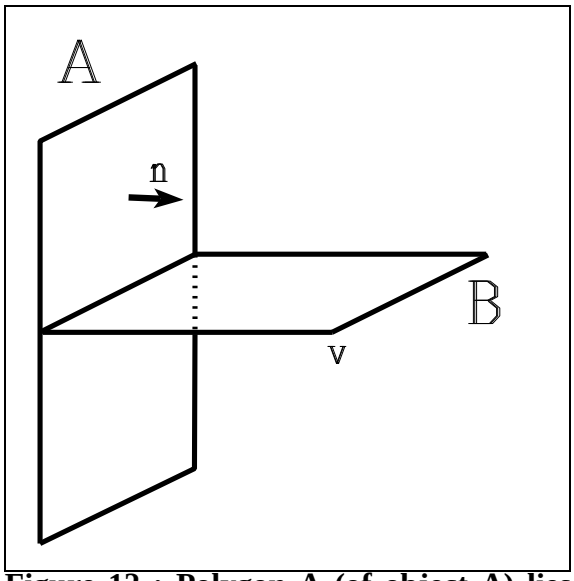

Figure 12 : Polygon A (of object A) lies on an edge of polygon B (of object B). n is the normal to polygon A. v is a vertex on polygon B, which is not on the intersecting edge.

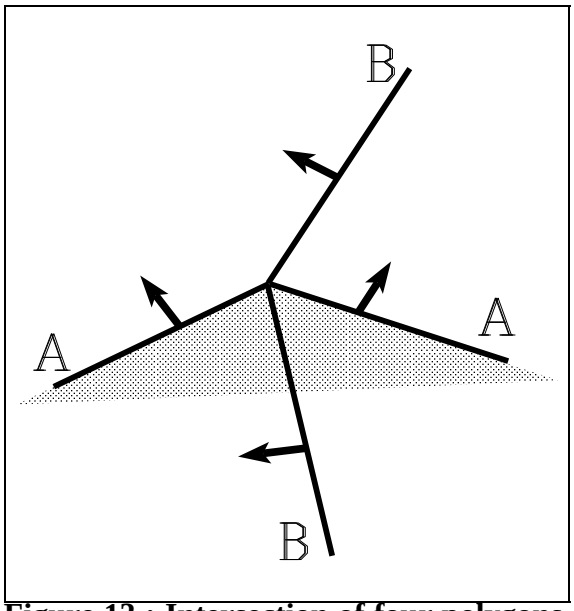

Figure 13 : Intersection of four polygons. The lines marked *A* represent polygons of object A, and those marked *B*, are from object B. The shaded area represents the inside of object A.

In the second case, all four polygons must be tested. The following steps should be used to determine if each polygon is inside or outside the other object (refer to Figure 13).

- Find the acute region between the two polygons of the other object (the region containing the acute angle between the polygons).
- Determine if this region is inside or outside the other object.
- Determine if the polygon being tested is inside or outside the acute region. It is inside iff it is on the appropriate side of both polygons of the other object.

The neighbourhood of the polygon can be deduced from the last two steps.

3.4.6. Decompose non-convex polygons

Although all polygons of the input objects are convex, the truncation of polygons in the fifth stage may result in non-convex polygons. These polygons must be decomposed into several convex polygons in order to produce a valid resultant object.

Several algorithms have been developed for decomposing non-convex polygons. The algorithm given below is a slight modification of the algorithm used in IRIT. An illustration of the algorithm is given in figure 10.

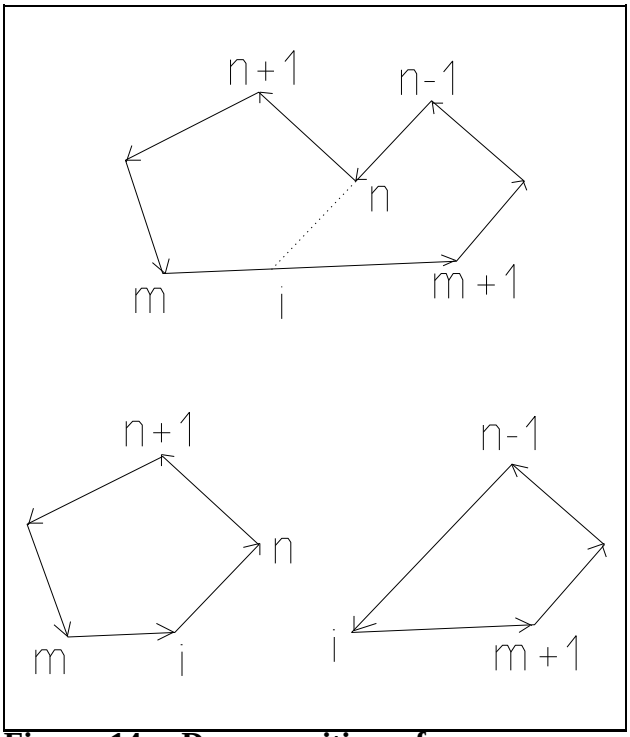

Figure 14 : Decomposition of a non-convex polygon.

- Traverse the polygon until a concave vertex (**n**) is found. If no concave vertex exists then the polygon is convex and no further decomposition needs to be done.
- Fire a ray from the previous vertex (**n-**$_1$) through the concave vertex (**n**), and find its closest intersection (**i**) with another line segment (**m, m+**$_1$).
- Remove the concave vertex by splitting the polygon along the ray. This split is performed by constructing the two polygons as follows :

  ..., m, i, n, n+$_1$, ...

  ..., n-$_1$, i, m+$_1$, ...
- Decompose the two polygons resulting from the last step.

3.5. Implementation

Most fast polygon rendering packages, including VR-386, use a very static storage structure for objects. This is done to optimize the rendering, translation, rotation and scaling of objects. The problem with storing objects statically is that their structure cannot be altered after creation. By the structure of an object, we refer to the number of polygons in the object, and the number of vertices in each polygon.

Algorithms for calculating the Boolean set operations require that many existing polygons are removed, truncated or split. It is therefore essential that objects be stored in a dynamic structure before the Boolean set operations are applied to them.

Because rendering speed is of utmost importance to VR systems, static structures are often used as the primary object storage. It is therefore necessary to transform objects from their conventional static storage to a dynamic storage before applying the Boolean operations. Furthermore, the object resulting from the operation must be transformed back into the static form.

Another consideration when implementing the Boolean operations is precision. In the quest for speed, the coordinates of the vertices of objects are often stored in the integer domain. Performing Boolean set operations, however, requires that vertices are highly precise. Coordinates should therefore be mapped into a real domain before applying the operations.

The Boolean set operations were successfully implemented for use in the virtual modelling system. The time taken to perform the operations was reduced to an average of under 0.2 seconds for a resultant object size of 130 polygons.

Chapter 4

Support for Head Mounted Display (HMD)

This chapter presents a system developed to provide support for the HMD. Various implementations are considered, and their performance evaluated.

4.1. Overall design

4.1.1. Hardware used

The HMD (illustrated in Figure 2 on page 8, and at the bottom of Figure 15) contains two LCD TV screens. Two VGA to PAL converters are used to convert the VGA output from a computer into a form that can be displayed by the TV screens. Because each computer can only output one VGA signal, at least two computers are required (one for each screen).

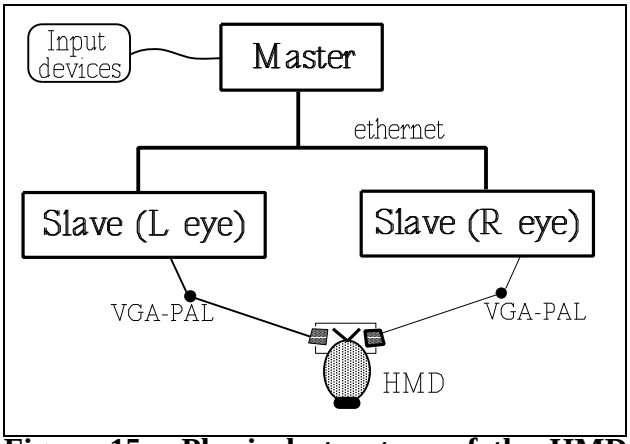

Figure 15 : Physical structure of the HMD system

The layout of the hardware used in the system presented in this chapter is illustrated in Figure 15. The system uses three computers, one master and two slaves. The master receives all input to the system. All processing, except for the image rendering, is done by the master. The rendering of the world is done by the slaves, which are connected to the master by ethernet cable.

4.1.2. Communication protocol

In order to render a view of the world, each slave needs to have an up to date copy of the visual components of the world. In addition, the position and orientation of the user must be known.
Any changes to the world must therefore be transmitted from the master to each slave.

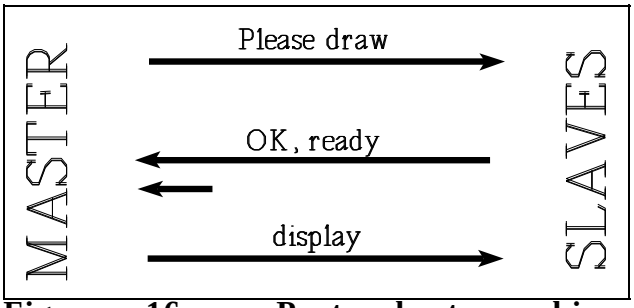

Figure 16 : Protocol to achieve synchronisation

In addition to keeping the slaves up to date, communication should also ensure synchronization of the slaves. In order to obtain this synchronization, the following protocol was adopted (refer to Figure 16).

- First, the master sends a *"Please draw"* message to the slaves.
- On receiving this message, each slave starts rendering a picture of the world (not visible to the user).
- Once each slave has completed rendering the world, it sends an *"OK, ready"* message back to the master.
- Once the master has received an *"OK, ready"* message from both slaves, it sends out a *"display"* message.
- On receiving the *"display"* message, each slave makes the rendered image visible to the user.
- Return to the first step.

4.2. Optimization issues

In this section, various implementations that adhere to the protocol mentioned in the previous section, are considered and compared.

4.2.1. Linear approach

This approach is illustrated in Figure 17.
The *"world update"* arrow represents all the communication required to update the world and user position.
The *"interact with world"* box represents all the work done by the master except for the communication. The *"render world"* box represents all work done by the slaves except for communication.

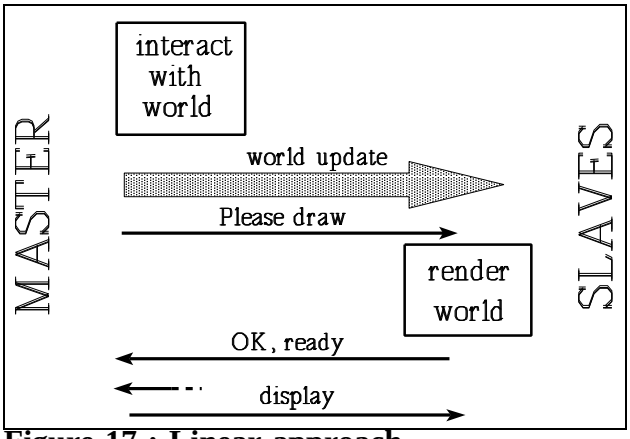

Figure 17 : Linear approach

In both this approach and the pipeline approach (covered in the next section), the master controls the system by performing a set sequence of steps. The slaves just receive messages from the master, and perform tasks determined by the messages.

The three possible message, and the slaves response to each is given below.

- If the message contains information about a change to the world, then the change is made to the slave copy of the world.
- If the message is *"Please draw"*, then the slave renders the world and returns an *"OK, ready"* message.
- If the message is *"display"*, then the slave displays the rendered image.

With the linear approach, the master performs the following steps in order :

- Read the input devices and perform any processing required on the world.
- Send a *"Please draw"* message to the slaves.
- Wait until both slaves have returned an *"OK, ready"* message.
- Send a *"display"* message.
- Send all information required to update the slaves copy of the world.
- Return to the first step.

Note that, with this approach, the master and slave never work simultaneously.

4.2.2. Pipeline approach

The pipeline approach is illustrated in Figure 18.

The only difference between this approach and the linear approach is the order in which the master performs its tasks.

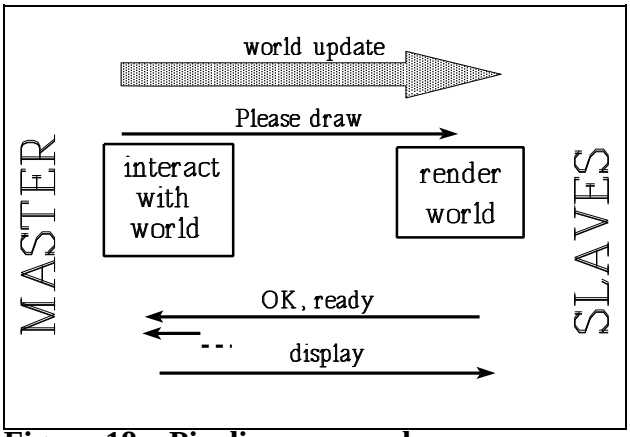

Figure 18 : Pipeline approach

With the pipeline approach, the master performs the following steps in order :

- Read the input devices and perform any processing required on the world.

- Wait until both slaves have returned an *"OK, ready"* message (to draw the previous version of the world).
- Send a *"display"* message to both slaves.
- Send all information required to update the slaves copy of the world.
- Return to the first step.

With this approach, at least part of the work done by the master and slaves is done at the same time.

4.2.3. Comparison of the Linear and Pipeline approaches

When evaluating the efficiency of a VR system, two main properties are considered. The first is the frame rate (the number of frames displayed per second). The second consideration is the latency. The latency is the time between when the user uses an input device, and when the result of using the device is visible to the user.

An input devices can be used at any time, but is only evaluated once per frame. The latency is therefore dependant upon the stage of the cycle that the system is in when the device is used. The best latency occurs when the input device is used just before the input is considered (i.e. at the start of the master's work). The worst latency occurs when the device is used just after input is considered.

For the rest of this section, Let
t_m = time taken by the master to read the input and perform all computations on the world.
t_s = time taken by each slave to render one frame.
t_c = time taken for communication between the master and slaves.

The time per frame, as well as the best and worst latency for each approach can then be calculated as follows :

Linear approach

Time per frame = $t_m + t_c + t_s$

Best Latency = $t_m + t_c + t_s$

Worst Latency = $2(t_m + t_c + t_s)$

Pipeline approach

Time per frame = $\max \{t_m, t_s\} + t_c$

Best Latency = $2(\max \{t_m, t_s\}) + t_c$

Worst Latency = $3(\max \{t_m, t_s\}) + 2t_c$

With the pipeline approach, the master and slaves work concurrently. After the master is finished, it waits for an *"OK, ready"* message from the slaves. This means that nothing else is done until both the master and slaves have finished.

The performance of the pipeline approach is, therefore, influenced by the ratio of the workload on the master to the workload on the slaves. A comparison of the linear and pipeline approach at the two extremes of this ratio is given in Table 2. Although $t_m=0$ (i.e. slave does much more work than master) was chosen for the unbalanced case, choosing $t_s=0$ yields a similar result. A summary of the comparison in table 2 is given in Table 3.

Table 2 : Comparison of the Linear and Pipeline approaches.

Workload		Time per Frame	Worst Latency	Best Latency
COMPLETELY BALANCED ($t_m=t_s$)	**Linear**	$t_c + 2t_s$	$2t_c + 4t_s$	$t_c + 2t_s$
	Pipeline	$t_c + t_s$	$2t_c + 3t_s$	$t_c + 2t_s$
COMPLETELY UNBALANCED ($t_m=0$)	**Linear**	$t_c + t_s$	$2t_c + 2t_s$	$t_c + t_s$
	Pipeline	$t_c + t_s$	$2t_c + 3t_s$	$t_c + 2t_s$

Table 3 : Summary of the comparison between the Linear and Pipeline approaches.

	Time per Frame	**Worst Latency**	**Best Latency**
Even Workload	Pipeline is best	Pipeline is best	equal
Uneven Workload	equal	Linear is best	Linear is best

Two main conclusions can be drawn from table 3. The first is that the frame rate is highest with the pipeline approach, whereas the latency is usually better with the linear approach. The second conclusion is that the pipeline approach performs better when there is an equal workload, and the linear approach performs best when there is an uneven load.

The adaptive parallelism approach (presented in the next section) is designed to take advantage of the second conclusion by switching between a linear and pipeline mode of operation.

4.2.4. Adaptive parallelism

The adaptive parallelism approach is a combination of the linear and pipeline approaches. With this approach, the workloads on the master and slaves are continually monitored. Whenever the load balance favours the linear approach, a linear mode is used. When the pipeline approach is favoured, a pipeline approach is used. The monitoring is done by timing all work, except communication, on each slave and on the master. The timing obtained on the slaves can be sent to the master with each *"OK, ready"* message. The master can then compare its workload with the workload of the slaves, and decide whether to use the linear or pipeline mode.

The system is controlled by executing the following sequence on the master.

1) Read the input devices and perform any processing required on the world.
2) Send all information required to update the slaves copy of the world.
3) If the master and slaves have a balanced workload (within a given tolerance), then go to step 7.
4) Wait for an *"OK, ready"* message from each slave.
5) Send a *"display"* message to both slaves.
6) return to step 1.

7) Read the input devices and perform any processing required on the world.
8) Wait for an *"OK, read"* message from each slave.
9) Send a *"display"* message.
10) Send all information required to update the slaves copy of the world.
11) If the master and slaves have an unbalanced workload, then go to step 3, otherwise go to step 7.

The first six steps correspond to the linear approach, and steps seven to eleven correspond to the pipeline approach.

4.3. Implementation

The HMD system was successfully implemented by modifying the VR-386 libraries to form a slave and a master version.

The communication layer, i.e. the routines to send and receive the various message via the ethernet, was written by Shaun Bangay.

The communication protocol presented earlier in this chapter can easily be modified to enable the use of **n** slaves, where $n \geq 1$. The only change required is that, instead of the master waiting for two *"OK, ready"* messages, it waits for **n** *"OK, ready"* messages. This extension, implemented with the HMD system, has proved very useful for demonstrations. It allows an active user to wear the HMD, and an audience to view what the user is doing, on other standard monitors. The extra displays also enable direct observation of user's interactions with a given VR environment, which helps to identify parts of the interface that need improvement.

4.4 Conclusion

All alterations made to the VR-386 libraries for the master are completely transparent to the VR-386 application interface. Any VR application using VR-386 can, therefore, be linked and run with the modified libraries without any modification. This system therefore makes the HMD available for further VR research at Rhodes University.

Chapter 5
Virtual Modelling Environment

5.1. Interacting with objects

The implemented system uses a Power Glove to enable the user to interact with objects in the virtual world. The interface is driven by several hand gestures (shown in Table 4). Note that only the thumb and first three fingers are used, as the Power Glove does not detect bend in the forth finger. By using these gestures, the user can select, rotate, scale and moved any object in the virtual world. The procedure used to accomplish each of these tasks is given below.

- **selection**. First, the "point" gesture is made and maintained. The desired object is then selected by poking it with the index finger. The object is automatically unselected when a different object is selected. The remaining tasks are all performed on the selected object.

- **rotation**. An object is rotated by using the "pinch" gesture. The user "pinches" a point in space and moves it around the object. The object is repeatedly rotated so that the same boundary point of the object remains on the line from the object centre to the index finger.

- **scaling**. The "double point" gesture is used for scaling an object. As soon as the gesture is recognized, the distance from the tip of the index finger to the centre of the object is calculated and stored. Each time the hand is moved (while the gesture is maintained), the original object is copied and scaled to form a new object. The scale factor is calculated by dividing the new distance (from the index finger to the canter of the object) by the initial distance. If the gesture is first made on the surface of the object, then the surface remains on the finger as the object is scaled.

- **translation**. The selected object is translated (moved) by making the "fist" gesture and then moving the hand. While the "fist" gesture is held, the object is attached to the hand, i.e. its position relative to the hand is maintained. Translating the hand therefore results in an equal translation of the object. The object is detached from the hand as soon as the "fist" gesture is no longer recognized.

The hand, constructed from the Power Glove input, is shown in Figure 19. At the time of this picture, the user was making the "fist" gesture, and moving the selected object (the globe near the hand).

Table 4 : Hand gestures used with the implemented system.

GESTURE	Thumb	Index finger	Middle finger	Ring finger
fist	bent	bent	bent	bent
point	either	straight	bent	bent
double point	either	straight	straight	bent
pinch	bent	bent	straight	straight
free	Any other combination.			

5.2. Navigation

Ideally, navigation should be done with the use of a head tracking device. Such a device would enable the user to change his or her position in the virtual world in the same way as he or she would in the real world. Because no suitable tracking device was available for this project, a compromise had to be made.

The implemented system uses a joystick to change the users position and orientation in the world. One problem with using a joystick is that movement is restricted mainly to the direction in which the user is facing. Another problem is that the joystick is a 2D device, which makes navigation in a 3D world less intuitive.

5.3. Menus

One of the main problems with existing modelling systems is that they rely on extensive menu systems. These menu systems are counter-intuitive and force the user to perform repetitive tasks.

By using a gesture driven interface such as the one mentioned above, the size of the menu system required can be substantially reduced.

By using a virtual reality interface, the menu system can also be made much more intuitive and effective. The approach taken in the implementation of the menu system is to make interaction with menus similar to interaction with conventional objects.

The menu in the implemented system consists of a menu frame, information objects and functional objects. All information and functional objects are attached to the menu frame. By selecting the menu frame, the user can rotate, scale or translate the entire menu as if it was a conventional object. The information objects are used only to convey information to the user, and can not be selected. Functional objects are objects that perform some function when selected. As soon as the function is performed, the object is unselected, making rotation, scaling and translation impossible.

An example of a menu used in the virtual modelling system is the menu for basic shape creation (seen on the right hand side of Figure 19). It consists of a menu frame and four functional objects representing a cube, cone, globe and cylinder respectively. When selected, each functional object produces a copy of its visual component, and places it in the user's hand.

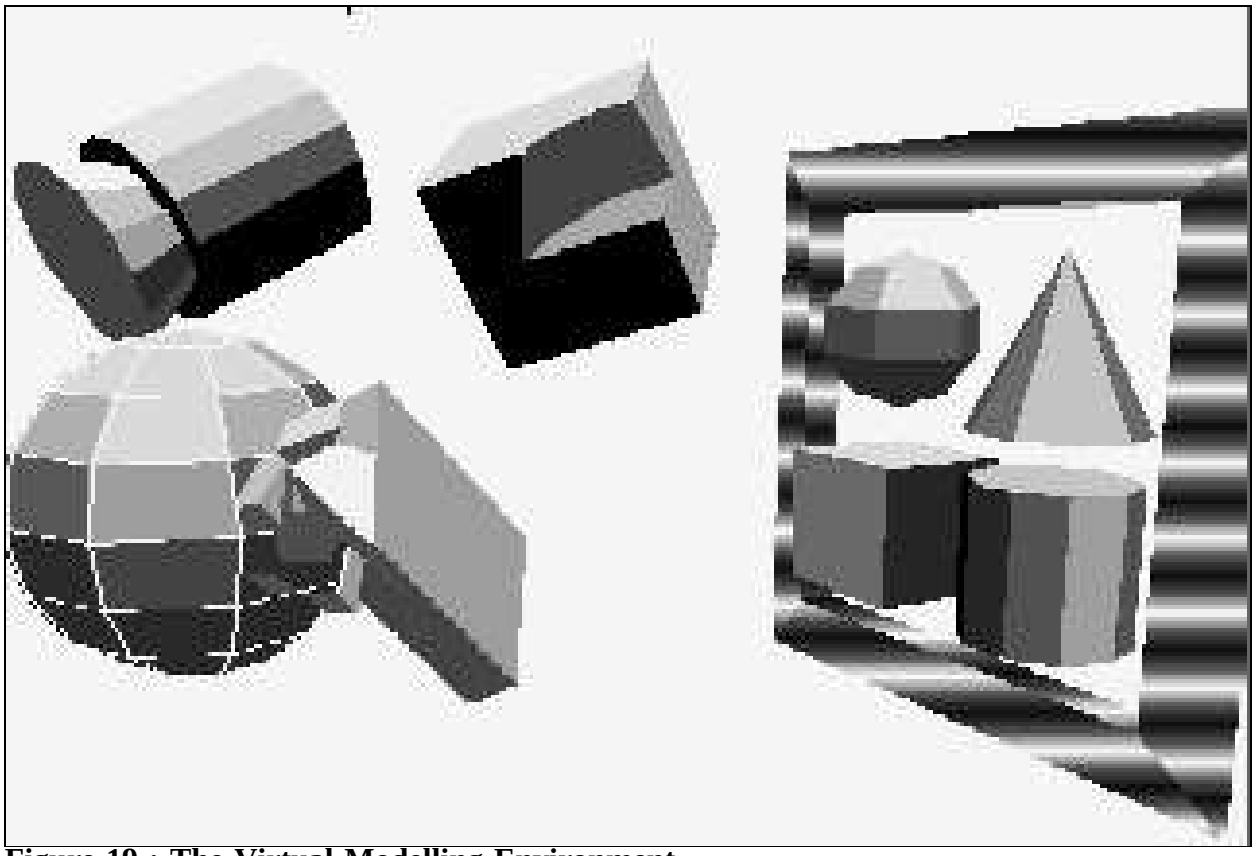

Figure 19 : The Virtual Modelling Environment.

5.4. Using modelling tools

The main modelling tools, i.e. those tools that use the Boolean set operations, were implemented using mode selection and collision detection. The selection of modelling modes can be done with a menu mentioned in the previous section (at the moment, modes are selected with the keyboard).

A modelling mode dictates what should happen when two objects collide, i.e. when one object (the tool) is moved into another object (the subject).

To define a modelling mode, we must specify four things :

- Which Boolean operation is to be performed on the two overlapping objects.
- Which objects (tool, subject and/or result) should be shown at each step of interaction.
- Which two objects should be kept after each step (to be used as the input for the next step).
- Which objects should remain once the interaction is complete, i.e. when the two object no longer overlap, or the tool is unselected (by selecting a different object).

Examples of five modelling modes are given in Table 5.
With the first three modes, the resultant object is only kept once interaction is finished. This allows the user to move an object into position and then perform the desired Boolean operation. With the last two modes, the object resulting from the Boolean operation at each step is used as an input object to the next step. The Boolean operations are therefore continually applied, directly modifying the shape of the resultant object.

It should also be possible to toggle the use of modelling modes on or off.

Table 5 : Examples of modelling modes.

MODES	Operation	Keep	Show	End result
union	$\cup$	subject, tool	result	result
intersection	$\cap$	subject, tool	result	result
difference	$\setminus$	subject, tool	result, tool	result
additive	$\cup$	result, tool	result	result, tool
subtractive	$\setminus$	result, tool	result, tool	result, tool

5.5. Conclusions

The approach taken when developing the virtual modelling environment was to keep the environment open and restrict the user as little as possible. In the resulting modelling system, no restrictions exist on the placement or relative size of objects. The user can, therefore, build a modelling environment around himself/herself that suites his/her modelling style and the modelling task to be performed.

The virtual modelling system was tested by over fifty people with a wide range of computer literacy. Most of the volunteers understood the operation of the system and the use of the gestures almost immediately. The actual execution of the tasks, e.g. the selection and fine positioning of objects, tended to take longer to master. The main factor hindering the performance of the users was found to be the instability of the Power Glove. Once the users were familiar with controlling the Power Glove, their performance in the modelling system was dramatically improved.

Chapter 6
Conclusions

6.1. Support for HMD

Support for the HMD was successfully implemented. The resulting system uses three computers. One computer acts as a master. It receives all the user input. It also performs all computations required by the VR application, except that of displaying the world. The other two computers act as slaves. They receive object and user information from the master. From this information, each slave renders an image of the world as viewed from the users left or right eye.

Various schemes for communication and synchronisation between the master and slaves were considered and compared. An adaptive parallelism approach was developed to optimize the efficiency of the system under all conditions. This approach involves continuously choosing between a linear and pipeline mode of operation. The performance of the resulting HMD system was found to be sufficient for use in most VR applications.

The system was implemented in such a way that all alterations to VR-386 are transparent to the application interface. The HMD can therefore be used by any VR-386 application program without any alteration to the source code.

6.2. Solid Modelling tools

Several modelling tools were evaluated and compared. The tools based on the Boolean set operations were selected for implementation.

Existing Boolean set operation routines (mentioned in section 3.4) were successfully modified to suit application in a virtual modelling environment. Several optimizations were also made to the algorithm and its implementation.

The resulting Boolean set operations proved fast enough for direct manipulation of relatively small objects. An average of under 0.2 seconds is taken for Boolean operations that form resultant objects of over 130 polygons.

6.3. Virtual Modelling Environment

A virtual modelling environment was implemented. It uses the Head Mounted Display for stereoscopic output, and the power glove for intuitive input. The system is designed to restrict the user as little as possible. In the implemented system, all components of the modelling environment can be arbitrarily placed. This enables the user to build the environment around him/herself in a way that suits the specific modelling task to be performed.

When designing the modelling environment, emphasis was also placed on the simplicity of user interaction. The resulting interface is easily used, even by people with a low level of computer literacy.

6.4. Possible extensions and future research

The virtual modelling system developed with this project can be extended in several ways. One extension is that of incorporating a head tracking device into the system.

At present, the modelling system focuses mainly on the first, rapid construction, stage of object construction. A second possible extension to the system is, therefore, implementing tools and modelling techniques specifically designed for work in the final, "touch up", stage of object modelling.

In addition to providing a virtual modelling system, which can be used as a base or starting point for future researchers, support is provided for the HMD.

This support makes the HMD available for further VR research at Rhodes University.

6.5. Concluding remarks

VR devices are continually being improved, and new devices are being developed. As these devices become available, new modelling techniques can be developed to take advantage of them. In addition, as computers become more powerful, modelling systems can afford to use modelling tools and techniques that were previously too computationally expensive.

The field of virtual modelling is, therefore, one in which the possibilities are rapidly expanding, and one which will support high level research for many years to come.

References

[Badouel 88] D. Badouel and G. Hégron. **"Set Operation Evaluation Using Boolean Octree"**. New Trends in Computer Graphics pp 275-287 Proceedings of CG International 1988

[Balaguer] Francis Balaguer and Angelo Mangili. **"Virtual Environments"**. Computer Graphics Laboratory. Swiss Federal Institute of Technology Lausanne.

[Bill 94] James R. Bill and Suresh K. Lodha. **"Computer Sculpting of Polygonal Models using Virtual Tools"**. July 1994

[Bricken 90] Wiliam Bricken. **"Virtual Reality : Directions of Growth, Notes from the Siggraph 1990 panel"**.

[Butterworth 92] Jeff Butterworth, Andrew Davidson, Stephen Hench and T.Marc Olano. **"3DM: A Three Dimensional Modeler Using a Head-Mounted Display"**. Symposium on Interactive 3D Graphics 1992.

[Foley 91] James Foley, Andries van Dam, Steven Feiner and John Hughes. Computer Graphics Principles and Practice. pp 535-539 **"Regularised Boolean set operations"**. 1991.

[Galyean 91] Tinsley A. Galyean and John F. Hughes. **"Sculpting: An Interactive Volumetric Modeling Technique"**. Computer Graphics, 25 (4), July 1991

[Putnam 86] L. Putnam and P. Subrahmanyam. **"Boolean Operations on n-Dimensional Objects"**. IEEE Computer Graphics and Applications 6 (6), June 1986 pp 43-51

[Requicha 78] A.A.G. Requitcha and R.B. Tilove. **"Mathematical Foundations of Constructive Solid Geometry : General Topology of Regular Closed Sets."** Production automation project 27, University of Rochester, New_York, march 1978.

[Rossignac 91] J.R.Rossignac. **"Through the Cracks of the Solid Modelling Milestone."** Eurographics 1991 pp 23-109. 1991

[Sproull 90] Robert F.Sproull. **"Parts of the Frontier are Hard to Move"**. Computer Graphics, 22 (2):9, March 1990. Symposium on Interactive 3D Graphics.