The Development of a Virtual Reality Simulator for certain Gastrointestinal Endoscopic Procedures

by

CHARLES CLAUDIUS MARAIS

DISSERTATION
PRESENTED FOR THE DEGREE OF
MAGISTER SCIENTIAE

IN THE FACULTY OF SCIENCE
IN THE DEPARTMENT OF COMPUTER SCIENCE AND INFORMATICS

AT THE
UNIVERSITY OF THE ORANGE FREE STATE
BLOEMFONTEIN
SOUTH AFRICA

SUPERVISOR: PROF. C.J. TOLMIE

MAY 1999

Acknowledgements

5DT <Fifth Dimension Technologies> - for financial and technical support as well as equipment supplied. Also for proposing the initial concept of this simulator.

Prof. C. Janse Tolmie – my promotor, for all his guidance and encouragement.

Dr. Johan P. du Plessis – for his guidance and encouragement.

Mr Paul Olckers – of 5DT <Fifth Dimension Technologies>, for his encouragement and valuable advice.

Dr. David M. Martin – of Atlanta South Gastroenterology Private Clinic, for making available valuable medical information and images.

Mr Rai Landau – of Protea Medical, distributors of Olympus endoscopic equipment, for making available a mechanically working gastroscope for this project.

Dr. Hennie de Klerk Grundlingh – of Universitas hospital, for valuable medical advice.

Dr. Jan van Zyl – of Universitas hospital, for valuable medical advice.

Mrs Martie Jacobs – for insight in some mathematical problems.

Mr Hanno Coetzer – for insight in some mathematical problems.

Mrs B.A. Janse van Rensburg – for linguistic help.

My family and friends – who supported and encouraged me.

My God – who guided me through these studies.

Contents

Chapter 1

Introduction

According to Sinroku Ashizawa, M.D. and Tsutomu Kidokoro, M.D. ([ASH70]), “Because of its facility of operation, safety, and superior photographic capability in providing objective detailed diagnosis, the gastrocamera has achieved wide acceptance in Japan and has become a routine diagnostic procedure in the management of gastric diseases.” The gastroscope, and all other endoscopes for that matter, has clearly become a very useful and important instrument for specialists and surgeons to make diagnoses. Currently reality-based (conventional) training for gastrointestinal endoscopic procedures is done in two steps. First the trainee watches how a specialist performs five to ten gastroscopies. Then the trainee performs between fifty and one hundred gastroscopies under supervision of a specialist.

A virtual reality (VR) simulator can be very useful for the training of medical trainees for the following reasons: Endoscopes are very expensive and trainees are normally not allowed to familiarise themselves too much with the controls of the endoscopes, other than performing a real procedure. In other words, the first time a trainee holds a gastroscope in his/her own hands, might be when performing a procedure on a live patient. This could be uncomfortable for the trainee as well as for the patient. If a trainee could practise on a simulator before he/she has to start with reality-based training on a real patient, the trainee will be better prepared, much more relaxed and will learn the real procedures much faster. The trainee may also practise certain difficult manoeuvres and therapeutic procedures after the reality-based training has started. New techniques and manoeuvres could also be developed using this simulator. Another advantage of such a simulator is of course that the simulator’s virtual patient is available twenty-four hours a day and will not complain, move or vomit.

In some countries it is becoming more and more important, due to extravagant lawsuits, to let patients sign informed consent forms before any medical procedure can be done on them. With such a simulator a scenario of what will be done inside the patient and how it will be done, can very easily be constructed. For example, showing a patient how a biopsy will be taken inside his/her stomach.

This thesis discusses the development of a VR simulator for the examination of the stomach. VR techniques are used in the construction of this computer-based system to enable the user to develop, practice and sharpen navigational and therapeutical skills as well as diagnostic skills. The system was specifically developed to run on a Pentium personal computer rather than a very expensive, special computer with enhanced graphic capabilities. Today's Pentium personal computers are powerful enough to handle enhanced graphics for VR applications and can cost up to ten times less than for example a Silicon Graphics computer. The system discussed in this thesis is ideal for teaching, training, simulation, patient briefings and research. It is very important to state that this VR simulator is not intended to *replace* reality-based training, but rather to *enhance* reality-based training by *preparing* a trainee on the simulator before he/she has to start reality-based training.

The system consists of a computer-based simulator, a 3-dimensional (3-D) tracking device, an endoscope and a life-size gastrointestinal model. See Figure 1.1 and Plate 1.1.

A Pentium personal computer is used to run the system. A normal endoscope is used with a hollow transparent life-size gastrointestinal model to provide maximum realism. The position and orientation of the front tip of the endoscope are tracked with the 3-D tracking device. This data is relayed to the computer, which then calculates and displays the appropriate image on the computer screen as realistically as possible. The calculated image closely resembles the image which would be seen with a real endoscope in a real patient. The image is continually updated in accordance with the movement of the endoscope/endocamera and the properties of the gastrointestinal model. Refer to the movie clips "**\Movie Clips\4- Gastroscope in Physical Model.avi**" and "**\Movie Clips\VR System\4- Orientation and Organ Windows.avi**" on the accompanied CD-ROM to get an idea of how the system works.

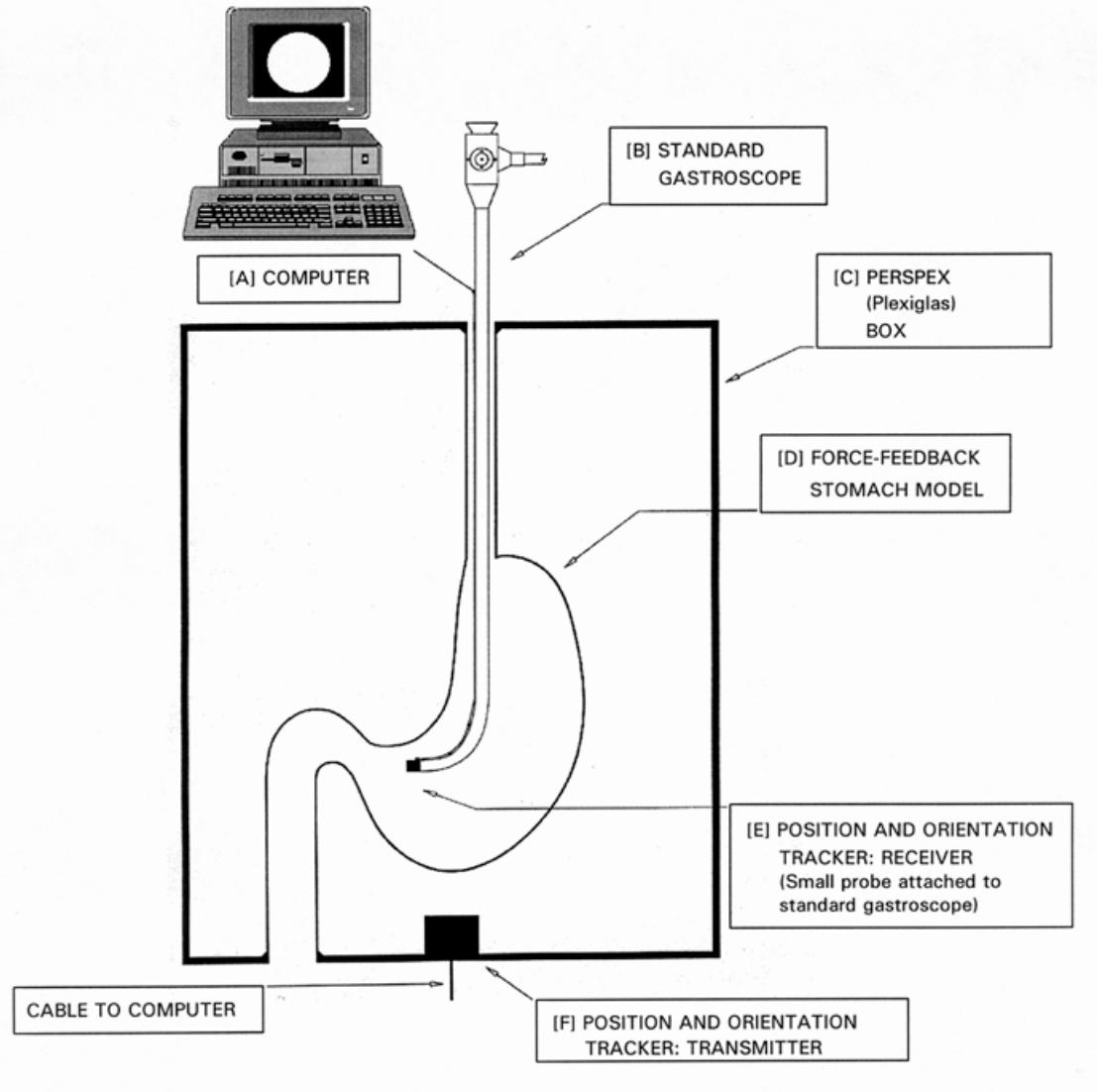

Figure 1.1 – Representation of the VR system.

The VR system contains a 3-D graphics computer model of the stomach, which is referred to as the virtual model. This virtual model is created to visually imitate a real stomach. Navigational and therapeutical skills may be practised by recalling or creating specific task lists for biopsy sampling, cauterisation and examining of certain regions in the stomach. Diagnostic skills may be practised by recalling specific case studies. These case studies may also be constructed by choosing abnormal gastrointestinal conditions from a database and applying them to the virtual model. The system can be operated in learn mode or test mode. Learn mode will show the trainee information and test mode will evaluate the trainee's diagnostic and navigational skills.

For this simulator to be successful, the simulator has to *realistically* simulate what a trainee would *see* and *feel* when doing a real procedure. Therefore the following problems originated from the development of this simulator: A computer graphic model of the stomach had to be generated with the same shape and size as a real stomach. Not only the shape and size had to be correct, but also the colour and texture inside the computer graphic model had to closely resemble that of a real stomach. Another problem was how to generate and insert abnormal conditions, like ulcers, inside the computer graphic model. Because so many gastroscopes are available and each has different specifications, like field of view, provision had to be made for setting up different gastroscopes to ensure maximum realism. Therapeutic tools also had to be generated to simulate procedures like taking biopsies. The last major problem was how to implement the system so that when the user touches the inside of the stomach with the gastroscope, it would feel as if he/she were touching the inside of a real stomach. In such a case the computer graphic model must also be deformed.

The layout of this thesis is as follows: Chapter 2 is a literature overview of endoscopic procedures, VR and work related to this project. Chapters 3 to 7 explain the design and development of this whole system. Chapter 3 contains a diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system. This diagram will be used to discuss Chapters 4 to 7. Chapter 8 was written independently from the previous chapters and should give the reader a very broad, but good overview of how this system works from the end user's point of view. Reading Chapter 8 before Chapters 3 to 7 might be helpful in understanding the whole system. Chapter 9 is a discussion of the results obtained with this simulator and Chapter 10 gives a conclusion to the thesis and discusses future work for the simulator. There are also two appendixes. Appendix A has all the algorithms or pseudo code discussed in this thesis and Appendix B contains a few pages of colour plates which will be referred to as Plate x.y. Several movie clips are also included in this thesis on the accompanied CD-ROM. The CD-ROM also contains all the colour plates. The structure of the CD-ROM is very easy to follow. The suggested way of viewing the colour plates and movie clips on the CD-ROM is to use the Windows 95 (or later version) explorer. Just double click on the colour plate or movie clip that needs to be viewed. The CD-ROM also contains the help file of the virtual reality system.

The system was developed for the Windows 95 operating system. The Watcom C++ version 10.6 compiler was used to compile the source code. The development of this system was done for the company 5DT <Fifth Dimension Technologies>. For the purpose of this thesis the system was developed to a satisfactory state and the research and development written down *only* up to that state of the system. The system described in this thesis still made use of a fibreglass physical model. The current system makes use of a silicon physical model, which feels much more realistic. 5DT plans to distribute the system commercially.

Chapter 2

Literature Overview

2.1 Introduction

This chapter will give a literature overview concerning this study. Section 2.2 will give an overview of gastrointestinal and endoscopic procedures, section 2.3 will give an overview of VR and section 2.4 will discuss previous and related work. It is important to have some background knowledge of gastrointestinal and endoscopic procedures, as well as understanding what is meant by VR, before reading through the following chapters. The sections in this chapter also explain some terms that will be used later in the thesis.

2.2 Gastrointestinal and Endoscopic Procedures

The term endoscopy refers to the examination or inspection of the internal organs, using a thin, flexible instrument inserted through body openings, such as the mouth or rectum. Organs that can be inspected by endoscopy include the esophagus, stomach, duodenum, colon, liver, pancreas, gall bladder, lungs, uterus and bladder. The liver, pancreas and gallbladder can only be seen by a combination of highly specialised endoscopic techniques performed by gastroenterologists using X-Ray guidance. This examination is called Endoscopic Retrograde Cholangio Pancreatography, or ERCP. ERCP is a highly technical procedure that is used to remove gallstones stuck between the gallbladder and intestines, as well as to aid in the diagnosis of many diseases of the pancreas [FOR93].

The most common endoscopes utilise optical fibres to relay the image from the endoscope tip to an eyepiece with a lens. Modern endoscopes utilise a video chip (miniature video camera at the endoscope tip) and strobe light to capture an image which is then displayed on a video monitor. This image can then be carefully examined for diagnoses. Endoscopy is also used to perform therapeutic procedures. Some therapeutic tools are biopsy forceps for removal of polyps and other abnormal growths, diagnostic needles, brushes, lasers, diathermy loops, balloons, baskets and stone crushers. Endoscopy is a very accurate aid in detecting inflammation, ulcers and tumours. It can also be used to detect early cancer and distinguish between benign and malignant conditions by taking small samples from suspicious areas with biopsy forceps. Endoscopic control of bleeding has reduced the need for transfusion, surgery and have minimised hospitalisation time for most patients. Generally these procedures are performed under light sedation, so that there is minimal discomfort [FOR93].

A bronchoscope is an endoscope used to examine the lungs. A gastroscope is an endoscope used to examine the esophagus, stomach and duodenum. A duodenoscope is an endoscope used to examine the duodenum, bile ducts and pancreas. A colonoscope is an endoscope used to examine the lower digestive tract, namely the colon. A cystoscope is an endoscope used to examine the bladder. A histeroscope is an endoscope used to examine the uterus.

Gastrointestinal endoscopy is a subspecialty of internal medicine that focuses on the intestinal tract. The first practical gastroscope was developed in Japan in 1953. The gastroscope has achieved wide acceptance and has become a routine diagnostic procedure in the management of gastric disease because of its safety and good photographic capability in providing objective detailed diagnoses [ASH70]. Figure 2.1 and Plate 2.1 illustrate what a generic gastroscope looks like and explain some of the controls on the gastroscope. Plate 2.2 shows how to hold a gastroscope and Plate 2.3 shows a magnification of the front tip of a gastroscope. Also refer to the movie clips "**\Movie Clips\1- The Gastroscope Controls.avi**" and "**\Movie Clips\2- The Gastroscope Controls (close up).avi**" that illustrate how the controls of a gastroscope work.

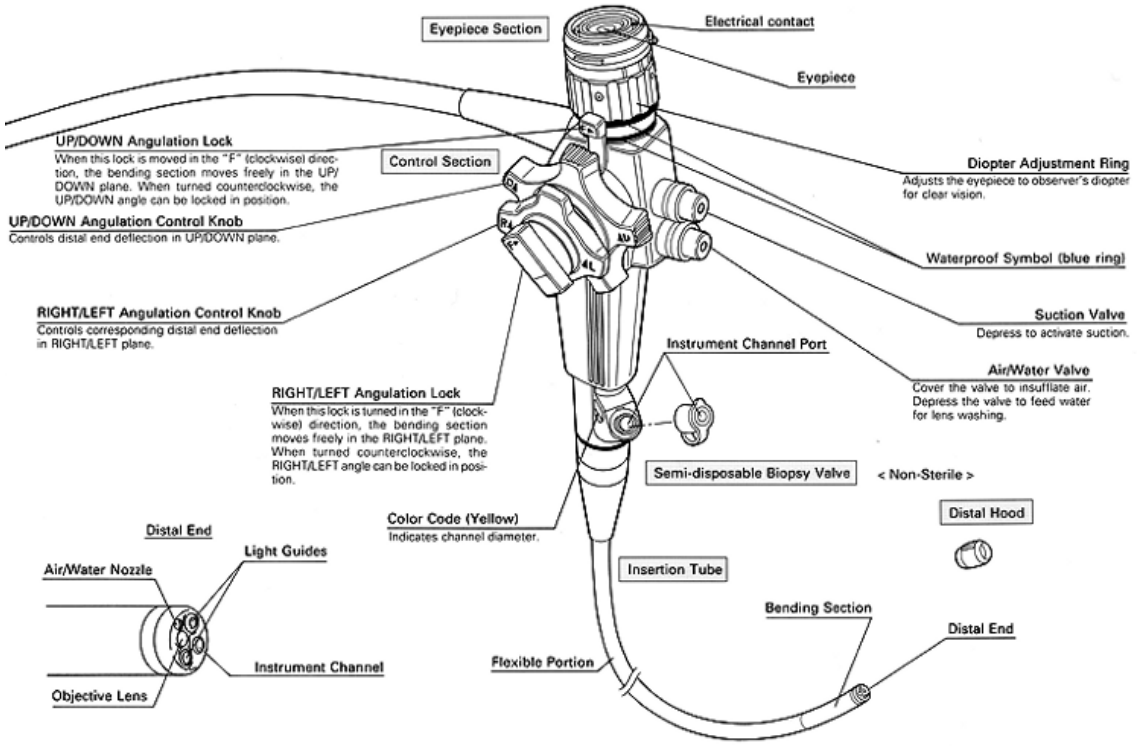

Figure 2.1 – An example of a generic gastroscope (Courtesy: Olympus).

Different kinds of gastroscopes exist. They can differ in the field of view, how far the tip can bend, the instrument channels and thickness of the scope. The field of view could range between 90 degrees and 140 degrees, but a typical gastroscope's field of view would be 120 degrees. The front tip can bend up, down, left and right. A typical gastroscope's up bending would be 210 degrees, its down bending 90 degrees and its left and right bending 100 degrees. Much slimmer gastroscopes exist specifically for paediatric patients. The

outer diameter of the insertion tube of a typical gastroscope could range from about 9 mm to 12 mm where the outer diameter of the insertion tube of a paediatric gastroscope could be about 5 mm [OESGIF].

2.3 Virtual Reality

This section will give a literature overview of VR. It will give a definition, discuss some VR devices, software and applications. Some of the VR devices discussed are data gloves, head-mounted displays and 3-D trackers. VR software and applications include modelling and rendering software, and simulators.

2.3.1 Definition

According to Burdea ([BUR94]), “Virtual Reality is a high-end user interface that involves real-time simulation and interactions through multiple sensorial channels. These sensorial modalities are visual, auditory, tactile, smell, etc.” VR differs from normal computer animation, because a three dimensional virtual world can be generated to *interact* with, in real-time, from a true perspective view of the virtual world. With normal computer animation, it is not possible to interact in real-time with the virtual world, and previously calculated views, that are stored sequentially, are used to display the animation sequence. The term virtual reality is therefore used to describe a system that generates real-time interactive visual, audio and/or haptic experiences. The senses of seeing, hearing and feeling can be used to experience and interact with these simulations of the real world or imaginary worlds.

What makes a VR application successful or not, is the number of human senses which are utilised, and how effective they are utilised. The sense of seeing is the most important. In order to utilise this sense effectively, realistic high-resolution graphical images have to be generated and displayed so that the visual effect is not jerky. If stereoscopic viewing is used, it is even more effective since better depth perception can be obtained. According to Larijani ([LAR93]), stereoscopic viewing is “imparting a 3-D effect; each eye receiving a slightly different image so that, when viewed together, what is seen appears to have depth”. The sense of hearing is probably the next most important. 3-D sound effects are

very effective, but stereo and even mono sound effects can still be used very effectively. Sound is important to a person's spatial awareness and is very effective when visual clues are minimal. Spatial sounds or surround-sound can be used to generate this spatial awareness. Spatial sounds can be described as "notes and tones appearing to emanate from different, varying distances; reproduced in virtual reality audio-spheres to enhance realism; type of surround-sound" [LAR93]. The sense of touch can also be utilised by using force feedback systems. Force feedback is the term used to describe a force that a person feels when touching or pushing against an object. Newton's third law of motion states that "If a particle exerts a force on a second particle, the second particle exerts an equal reactive force in the opposite direction" [VIN95]. The sense of taste and smell is very difficult to simulate, but these are not very critical senses to utilise and will probably not be utilised widely in the near future. Some VR applications are already utilising the sense of smell [BUR94].

2.3.2 Virtual Reality Devices

Apart from a computer and the software that generates the images of the virtual world, a number of physical input and output devices can be used to enhance the interaction with the virtual world. The head-mounted display or HMD and data glove are probably the two most familiar devices. The following devices will be discussed: display devices, data gloves, tracking devices and body suits.

2.3.2.1 Display Devices

Several display devices can be used with a VR system. Some of the more known devices are the computer monitor and the head-mounted display.

Computer Monitor

Most VR software applications make use of the computer monitor. All generated images of the virtual world are displayed on the screen, which makes the application accessible to more computer users because no special equipment is needed. The disadvantage is that the user cannot implement stereoscopic imaging.

The system developed during the writing of this thesis uses the computer monitor for a display device. This display device works very effectively since the output of a real videoscope is seen on a monitor which closely resembles a computer monitor. The more realistic the whole simulation can be presented to the user, the more successful the simulator should be.

Filter Glasses

Filter glasses are a cheap way of utilising stereoscopic imaging. It is used with a computer screen on which the stereoscopic images are generated. Filter glasses are basically a pair of glasses with two filters of different complementary colours (red, green or blue) for lenses, or different polarisation. For example, a filter glass can be made using a red filter for the left eye and a green filter for the right eye. Two images need to be generated for stereoscopic imaging, one image for the left eye and one for the right eye. With filter glasses the two images are placed on top of one another. The images have to be generated so that when looking through the red filter, only the left eye's image can be seen, and visa versa [VIN95]. The biggest advantage of filter glasses is that they are very cheap to manufacture. Another is that it is a multi-user device, which means that each person in a room can wear a set of glasses and all can look at the same images on the same display. It is not necessary for each person to have his/her own display. The disadvantage of filter glasses is that it is not that effective, since the two images have to be placed on top of one another, which could cause a blurry or "ghost" image, even when using the filter glasses [JAC94].

Another technique uses polarisation lenses to filter image pairs. A polarisation lens blocks light travelling in a horizontal or vertical plane, depending on the lens. The horizontal data for one image is displayed on the screen's upper half, while the vertical data for the same image is displayed on the lower half. The data is merged by a beam splitter and viewed with a pair of polarised filter glasses. A disadvantage of this approach is that the screen's vertical resolution is cut in half. Filter glasses also do not have sound or tracking capabilities. [JAC94]

LCD Shutter Glasses

Liquid crystal display (LCD) shutter glasses are more expensive than filter glasses, but much cheaper than HMDs. LCD shutter glasses work similar to filter glasses because the user must also watch a monitor through the glasses. As with the filter glasses, two separate images need to be generated, but with LCD shutter glasses, the two images are not placed on top of one another. The glasses are basically two LCD shutters which can open and close at a very high rate. When wearing the glasses, each eye has its own LCD shutter. If the left shutter is open, the right shutter is closed, and visa versa. Images have to be generated on the computer monitor so that one frame displays the left eye image and the next frame displays the right eye image. A synchroniser is used to synchronise the LCD shutters with the computer monitor to ensure that each eye only gets its image. If the refresh rate of the monitor is too slow, or the shutters cannot open and close fast enough, the image will appear to flicker. The advantages of LCD shutter glasses are that they are quite inexpensive and very clear stereoscopic images can be experienced with them. The shutter glasses can also be used in a multi-user environment. Compared to HMDs, a person can work much longer with LCD shutter glasses before he/she tires since they are not so big and uncomfortable to wear [BRO94]. The biggest disadvantage of shutter glasses is that because the whole screen is used to generate a single image for one eye, generating one stereo image requires the generation of two “whole” screens. Therefore, the image quality is very good, but the frame rate is cut in half. As with filter glasses, the user can see static stereoscopic scenes because regardless of the user’s viewing angle, the image will be displayed the same on the screen. Normally LCD shutter glasses do not have sound or tracking capabilities, but this could be added, as long as the wearer looks at the computer screen [JAC94].

Head-Mounted Displays (HMDs)

A head-mounted display has two very small video output screens placed very close to each eye. Some HMDs use very small cathode ray tubes (CRTs), but the most commonly used are LCDs. Each eye looks into a lens that magnifies the little screen so that the image can fill as much as possible of the eye’s field of view. The higher the quality, the higher resolution and wider field of view can be experienced with these screens. Because each eye has its own screen, stereoscopic imaging can be used to generate different images for each eye. Most HMDs have three-degrees-of-freedom tracking devices which are used to track

the orientation of the HMD [BUR94]. Tracking devices will be discussed later in this chapter. Wearing a HMD cuts the user off from the real world, since the view of the real world is mostly blocked. Most HMDs also have earphones that dampen noise from the real world and let the user hear 3-D sound effects. Some HMDs also have a microphone to use with voice input applications. The advantages of HMDs are that they can produce a very realistic virtual world by using stereoscopic imaging and 3-D sound effects. The main disadvantage is that they are (to date) very expensive. Some HMDs are also very heavy and uncomfortable to wear [BUR94].

2.3.2.2 Data Gloves

To interact with a conventional software application, a user normally uses a mouse or keyboard. These input devices are not really sufficient for interacting with 3-D worlds. 3-D track balls and probes can be used as 3-D input devices, but a much more intuitive way of interaction is with a data glove. A 3-D data glove is an input device that fits like a normal glove on the user's hand. Sensors are used to measure the finger joint angles. Some gloves also measure the angle at the wrist. This data can then be used as input in the virtual world. Other applications could be to play virtual music instruments, sign language or simulation of a 2-D pointing device [BUR94].

Different ways of measuring the bending of the fingers are used. Some use sensors that are made of a double layer of conductive ink with carbon particles. If the sensor stretches, the distance between the conductive carbon particles increases, causing an increase in the resistivity of the sensor. This resistivity data is transformed into joint angle data through calibration. The Mattel Power Glove uses this technique [JAC94].

Another method is to use thin electrical strain gauges. The joint angles are measured by a change of resistance in a pair of strain gauges. This change in resistance produces a change in voltage, which can be measured and used for calibration. The CyberGlove uses this technique [BUR94].

Optical fibres can also be used. Each joint has a fibre loop. On one end, a light source is connected and on the other end a phototransistor is connected. As a joint is bent, the index

of light refraction will change so that the joint angle can be measured by calibrating the intensity of the return light at the phototransistor. The 5DT Data Glove uses this technique. The 5DT Wireless Data Glove works with a high-speed wireless link [EXP98d].

2.3.2.3 Tracking Devices

3-D Tracking devices are also very important spatial input devices. These devices measure 3-D position (x, y, z) and orientation (yaw, pitch, roll) according to a specific origin. Figure 2.2 illustrates a right-handed coordinate system. Different methods of tracking are used, like ultrasound, electromagnetic, infra-red and video capturing. These will be discussed later in this chapter. With most of the tracking devices, a transmitter is used to transmit certain data and a receiver is used to receive this transmitted information to calculate position and orientation. Tracking devices are categorised according to the amount of information about position and orientation obtained. To measure 3-D position and orientation completely, a six degrees of freedom, or 6 DOF, tracking device would be needed. This means that a 3-D position, typically using x-, y- and z-coordinates, can be measured and that the rotation about the x-, y- and z-axis can be measured. For a right-handed axial system, rotation about the y-axis is also called “yaw”, rotation about the x-axis is called “pitch” and rotation about the z-axis is called “roll” [VIN95].

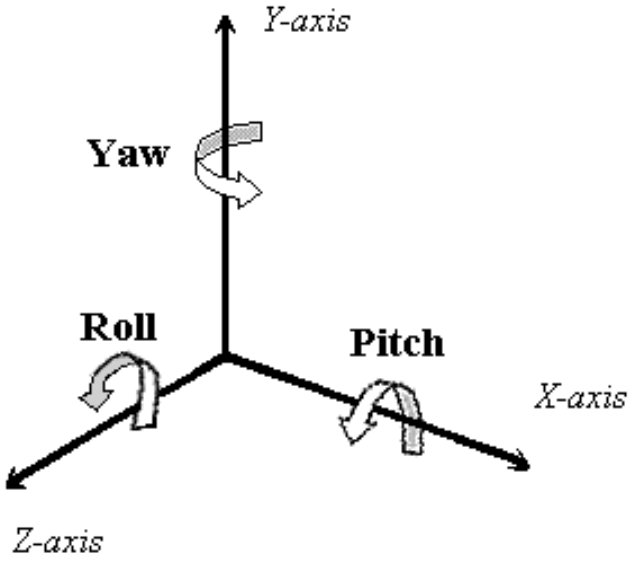

Figure 2.2 – Rotations of a right-handed system.

Tilt Sensors

Tilt sensors are used to measure orientation. Most data gloves have tilt sensors that can be used to measure the pitch and roll to calculate the orientation of the hand. This is called two degrees of freedom, because two degrees of orientation are measured. A tilt sensor is basically a device with liquid inside and a fixed piece of metal or any material that conducts electricity. As the sensor is tilted, the liquid inside the sensor tilts in the opposite direction and the current can be measured. Tilt sensors are commonly used in baseless joysticks and data gloves. Tilt sensors are normally used with an electronic compass (flux gate) which measures orientation in the yaw direction.

Ultrasound Tracking

An ultrasound tracking device uses three ultrasound speakers mounted on a fixed triangular frame as its transmitter. The receiver is a set of three microphones mounted on a smaller triangular frame. Since the speed of sound is known, the distances from each speaker to each microphone can be calculated and by using triangulation methods, the position and orientation of the receiver relative to the transmitter can be calculated. The disadvantage of this technique is that a direct line-of-sight is required between the transmitter and receiver, since nine distances need to be measured using the known speed of sound. This means that no objects should obstruct the line between the transmitter and the receiver that can deflect or absorb the acoustic waves. The quality of the transmitter and receiver will also directly influence the accuracy of the calculated 3-D position. Using ultrasound waves also have the disadvantage that external sound waves can easily interfere with the tracking process [BUR94]. Refer to Figure 2.3 for the functioning of an ultrasound tracking device.

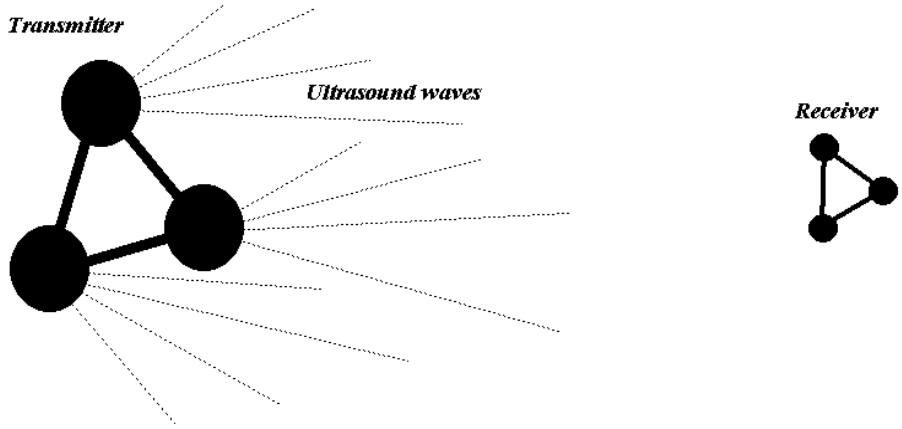

Figure 2.3 – Shows how ultrasound waves are generated by the speakers that form the transmitter. The receiver is a set of microphones.

Infra-red Tracking

Infra-red tracking devices work almost in the same manner as ultrasound tracking devices. A set of three light-emitting diodes (LEDs) are used with an infra-red camera. The infra-red tracking device functions are illustrated in Figure 2.4 and Figure 2.5. Figure 2.5 shows how it can be determined if an object is coming closer or going further away, if it rotates around its x-axis and/or y-axis. The advantage of this system is that it has very low interference, so the tracking can be very accurate. The disadvantage is that it is also a line-of-sight tracking device.

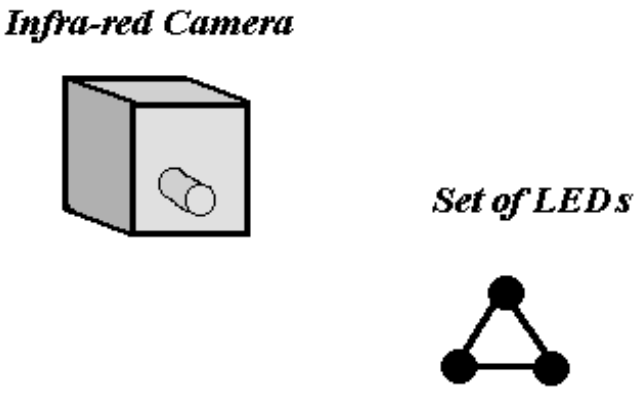

Figure 2.4 – Shows how infra-red tracking works with the infra-red camera that monitors the set of LEDs. See Figure 2.5 for some examples.

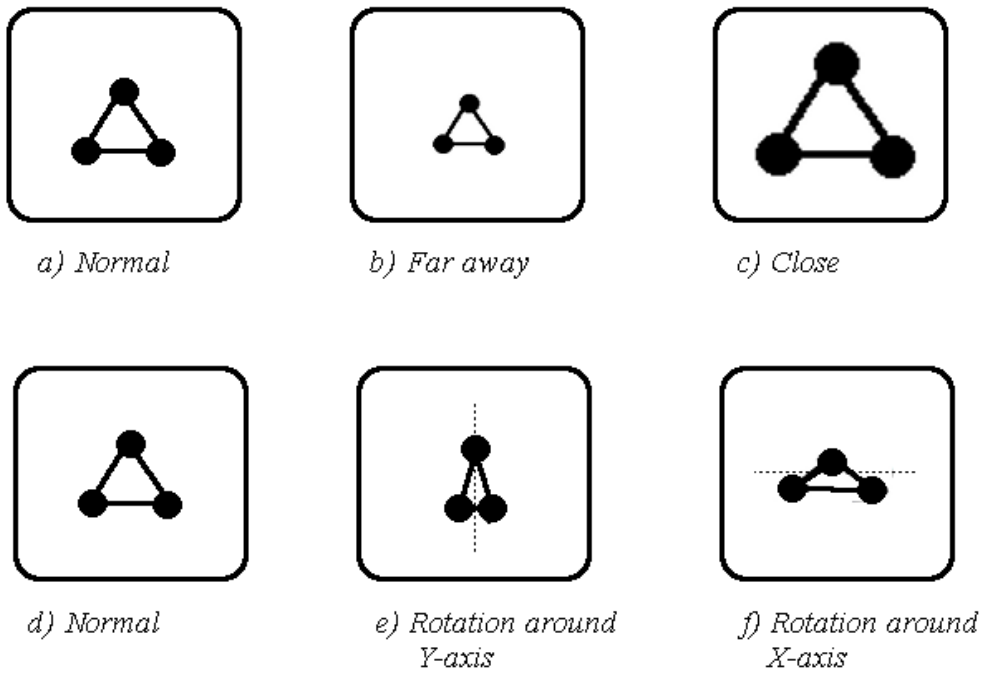

Figure 2.5 – Shows different infra-red camera views.

Electromagnetic Tracking

The electromagnetic tracking device uses electromagnetic fields generated by three stationary orthogonal coils in the transmitter and three coils in the receiver, to calculate a 3-D position and orientation of the receiver relative to the transmitter. These tracking devices are very popular, since they are very accurate and the transmitter and the receiver do not need to be in a line of sight. The disadvantage of this tracking device is that metal objects near the transmitter or receiver influence the readings of the tracking device. Also, electromagnetic fields are limited to distance, so if the receiver is too far away from the transmitter, the readings could be less accurate. These tracking devices are unfortunately also very expensive [BUR94]. Plate 2.4 shows the electromagnetic 3-D tracker used by this simulator.

Video Capturing for Tracking

Stereoscopic analysis of the correlation of pixels common to two images seen by two cameras can also be used as a tracking device. Basically, three light sources, fixed relative to one another, are captured with two video cameras with a fixed distance between them. Each camera has a different image of the three light sources and by analysing these two images, position and orientation can be calculated. Figure 2.6 illustrates how this can be

done. The disadvantage of this system is that because of using light sources, this system is also an in-line-of-sight system. The advantage of this system is that it is currently one of the most accurate tracking devices and has been used in the field of cinematography to capture complex body movements of an actor.

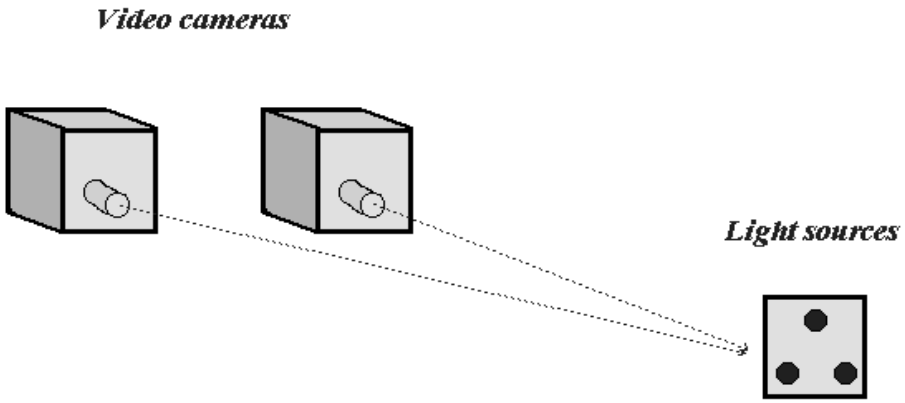

Figure 2.6 – Shows how video capturing can be used for tracking.

2.3.2.4 Body Suits

According to Larijani ([LAR93]), "A body suit is basically a customized dataglove for the whole body." Fibre optics are used to measure body movements. About 20 sensors could monitor all major body joints. 3-D tracking devices can be used for tracking of spatial movement, such as moving forward. A HMD can also be used very effectively with the body suit. Trying to move around in a suit that has a lot of data wires attached to it, is very difficult and some research has been done on "walking pods". Expensive body suits do exist that utilise radio frequencies rather than data cables to transmit data to the VR system [LAR93].

2.3.3 Virtual Reality Software

For any VR application to be successful, the VR hardware, VR software and user interface (hardware and software) must be integrated successfully. The previous section discussed most of the VR hardware. The term VR software refers to software modelling tools for building virtual worlds, simulating physical behaviours in virtual worlds and rendering software for representing the virtual worlds. According to Ferraro ([FER96]), rendering is "the process of creating a 2-D image from a particular viewpoint of a 3-D scene." This

section will briefly explain how the images that are seen in a HMD or on a computer monitor are generated.

2.3.3.1 Modelling Software

Software modelling tools, like Autodesk 3-D Studio and Caligari Corporation's TrueSpace, can be used to construct computer graphic models. A computer graphic model is a graphical representation of a real-life or abstract object on a computer. Both the model's shape and the material by which it is made can be described so that it can be displayed as realistically as possible. Models of computer aided design (CAD) applications can also be used in VR applications. Some other modelling software applications follow: "A 3-D computer model creation technology from Synthonics Technology lets PC users create and control models" [EXP98c]. "The HoloSketch VR Sketching System is a 3-D geometry creation and manipulation tool that gives non-programmers the ability to create highly accurate virtual objects by editing 2D projections of 3-D objects" [DEE96]. "The Silly Space modelling application lets the user manipulate implicit surfaces in a real-time 3-D environment" [LUS97].

2.3.3.2 Rendering Software

According to Watt ([WAT96]), "Rendering is the process of converting 3-D geometric descriptions of graphical objects into 2-D image plane representations that look real". Rendering software, like Criterion's Renderware and Microsoft's DirectX, is used to do the actual drawing of the images that are seen. A 2-D image is created from a particular viewpoint of a virtual world. The term virtual world refers to the virtual 3-D space where 3-D objects can be added and simulated. A virtual world is also known as a virtual scene. 3-D objects can be moved around in the virtual world and be simulated to interact with other objects. Apart from objects, virtual light sources and virtual cameras can also be added to the virtual world. Light sources are used to illuminate the objects in the scene and the cameras are used to define the 2-D image of the scene from a specific viewpoint [FER96].

The rendering process can be compared to a dark room in which objects can be placed. Although objects can be placed in the room, they will not be visible without a light source.

More than one light source can be used and each light source can be of a different colour. Several kinds of light sources can be used, for example, a directional light that shines light parallel to a direction vector or a point source that can be compared to a physical light bulb that shines light in all directions from a specific point. The objects are illuminated by the light source(s). To get a 2-D picture of the 3-D room from a specific viewpoint, a real camera can be used. Therefore, in a virtual scene, a virtual camera can be positioned in the scene and be used to generate 2-D images of the 3-D scene.

The system developed during the writing of this thesis uses Renderware from Criterion as rendering software. Renderware is a set of API functions that can be used with Microsoft Visual Basic, Microsoft Visual C++ or Watcom C++.

2.3.4 Virtual Reality Applications

Virtual reality is used in a number of fields, like entertainment, engineering, science, medicine and training. Useful applications are for training to work in dangerous environments and training of once-off scenarios. Dangerous environments are, for example medical surgery, mining, military, nuclear, chemical and high voltage yards. With VR these environments can be created as virtual worlds, simulating the properties of the main objects in that world. A person can then practise certain tasks or situations in the virtual world without the danger of being injured or harming somebody else. A once-off scenario is a situation where a person gets only one chance to perform or complete a certain task. A very good example of a once-off scenario is the repair of the Hubble telescope, which took place in December 1993. Astronauts had to replace the telescope's defective panels with new ones from the space shuttle's cargo bay. It costs millions of dollars for a single launch into space, therefore the astronauts had to repair the telescope the first time [BUR94]. Other examples could be the cutting of precious stones and cosmetic surgery.

The following sections will discuss a number of fields where VR is used, from entertainment such as computer games and movies, to medicine such as surgical simulators.

2.3.4.1 Entertainment

Entertainment is currently the largest market for VR and was the driving force of early VR technology. The first large-scale VR entertainment system was the “BattleTech Center” that opened in Chicago in August 1990 [BUR94]. This is a VR cockpit-based, team play game and the theme is a futuristic war fought by human-controlled robots. These cockpit-based virtual reality experiences are very popular. Another game is CyberTron, manufactured by StrayLight Corporation, which is an immersive VR game. The player wears a HMD while standing inside a gyro mechanism. The gyro mechanism moves in harmony with the player’s body weight and inertia. Players “fly” through obstacles, tunnels and mazes, while facing clever virtual opponents [VIN95].

With computer technology that can generate photo realistic images, the film industry is evolving with VR and the arts and technologies are merging [LAR93]. Examples of some films that made use of VR techniques are “Lawnmower Man”, “Terminator 2”, “Robocop 2”, “Toy Story” and “Lost in Space” [COM98]. The cost of VR hardware and software kept VR out of the home entertainment market for very long, but some lower end devices are available. Cheaper devices such as the 5DT data glove and the Virtual i-Glasses HMD are some of the VR devices that are available for home entertainment.

2.3.4.2 Business

Businesses are attracted by VR since it is a great advertising tool. VR can also be a useful tool in financial decisions. Decision-support systems can use VR visualisation techniques to clarify or simplify complex data so that people at a business can make informed decisions. Stock trading and currency exchange visualisation can also be made much easier to understand using VR techniques. An example of a stock trading program is the “Capri VR Trading tool” developed by Maxus. A stock market 3-D visualisation program called “vrTrader” was also introduced by Avatar Partners in 1993 [BUR94].

2.3.4.3 Engineering

The engineering community has successfully applied computer-aided design (CAD) techniques for design and manufacturing processes. Refer to section 2.3.3.1 for more detail about modelling software. Some everyday activities of engineering are preparing 2-D

schematics of wiring diagrams and floor layouts. 3-D CAD systems have been used very successfully for a number of years to design almost anything from a screw to an oilrig. According to Vince ([VIN95]), “The computer has now become the most powerful design tool ever created.” The reason for this is not just because of its speed and ability to display images, but because it is very flexible to change its role. Text can be processed, documents can be organised, line drawings can be developed and 3-D objects can be constructed. VR techniques allow engineers to design, inspect, assemble and test objects in a virtual environment. The designed objects can be subjected to all sorts of tests before the real objects are manufactured [VIN95]. Some examples of where VR techniques are used in the engineering industry follow.

Aero engines are designed very carefully to withstand incredible forces during operation. They must also function in all types of weather and over incredible ranges of temperature and atmospheric conditions. These engines are designed with the latest CAD systems, but full-size prototypes still play an important role in the investigation of all aspects of servicing. For safety reasons the engines are regularly removed from the aircraft for servicing and then replaced again. Rolls-Royce uses ComputerVision’s CADDS4X system for aero engine design and uses physical mock-ups for maintenance issues. To build these mock-up engines is very expensive. In 1992, Rolls-Royce approached Advanced Robotics Research Laboratory (ARRL) to undertake a feasibility study to see whether physical mock-up models could be replaced by VR technology. Using ARRL’s VR platform, Division’s SuperVision system, the feasibility study resulted in a successful demonstration and work continues [VIN95].

Yamaha Motor Corporation has installed Cray J916se, which is a supercomputer, to facilitate the design and testing of large-scale complex CAD models prior to production [COM97a].

Civil engineers at TranSystems are using software products running on Bentley Africa’s MicroStation to upgrade the signing at the airport coming in and out and around the terminal at Kansas City International Airport in Missouri. Some big changes, like new illuminated signs providing updated information, pavement markings and guard rails, will

be involved. Simulating and testing these changes using VR is much cheaper than actually changing the signs and then finding out which signs would work better [COM97b].

The San Francisco Bay Bridge was severely damaged in the 1989 Loma Prieta earthquake. Three new designs for the bridge have been recreated digitally using powerful new 3-D urban simulation software to make the Bay Bridge earthquake safe. With these simulations, developers and city planners can interactively visualise highways, airports and cities by using existing images, land and environment data [COM97c].

Power plant engineers are using virtual manufacturing software to plan the process of cleaning up the contaminated Chernobyl Nuclear Power Plant in the former Soviet Union. Software from Tecnomatix Technologies will be used to create a virtual model of Chernobyl. This model will be used to simulate and verify the tasks which robots must perform to clean up the plant and disassemble it [EXP98a].

The above examples clearly show how useful VR can be applied in the engineering industry. It can help save lives and money by designing and testing products in a virtual environment before the real products are manufactured. In some cases, like with aero engines, prototypes of the product must still be manufactured for safety reasons.

2.3.4.4 Science

Virtual reality can be a very useful visualisation tool in science. Data sets from the worlds of neuroscience, cartography, remote sensing, archaeology, molecular modelling, medicine and oceanography can be interpreted using visualisation techniques. According to Bryson ([BRY96]), “Scientific visualisation is the use of computer graphics to create visual images that aid in the understanding of complex numerical representations of scientific concepts or results.” These results might be the output of *simulations* like computational fluid dynamics or molecular modelling, *recorded data* like geological or astronomical applications, or *constructed shapes* like visualisation of topological arguments. Applying VR to scientific visualisation provides a real-time intuitive interface for exploring data. The main difference between scientific visualisation applications using VR and other VR applications is that scientific visualisation is oriented toward the informative display of

abstract concepts, as opposed to an attempt to realistically represent objects in the real world [BRY96]. Some examples of scientific VR applications follow.

Researchers in the field of computational neuroscience use simulation models of single neurons or networks to discover how the nervous system works. The research group at the University of Illinois in Chicago is investigating VR as a suitable visualisation tool. Their conclusion was that “head tracking and stereo images create a credible 3-D virtual space that simplifies the interpretation and understanding of very complex data sets” [VIN95].

The visualisation and manipulation of virtual molecular structures are of particular interest and use to chemists and biochemists. A VR approach to molecular modelling is for a scientist to interact directly with a graphical representation of a molecule [VIN95].

NASA’s Jet Propulsion Laboratory is using VR software to enable its scientists to collaborate in real-time over a VR network with colleagues at other facilities. Muse Technologies’ Continuum software lets team members in different geographic locations explore similar multi-sensory environments [EXP98b].

These examples show how useful VR can be applied as a visualisation tool in science. It can help scientists to explore abstract data by using informative visualisation displays and molecular modelling can also be done interactively.

2.3.4.5 Education and Training

Immersion and interactivity improve user learning and could improve knowledge retention and student motivation. Since VR is a relatively new field, not many studies have been done to measure its benefits as a teaching tool [BUR94, LAR93].

In 1991 and 1992, the first such studies were done at the University of Washington HIT Laboratory Summer School, and at the West Denton High School in Newcastle, UK. The subjects used for these two projects were between 13 and 15 years old. Although these first studies were aimed at learning *about* VR rather than learning to *use* VR, the majority of students were overwhelmingly enthusiastic about this new teaching tool and two thirds

even preferred exploring VR to watching television. Shephered School in Nottingham, UK, the largest school in Great Britain for children with severe learning difficulties, also explored the educational benefits of VR. The school used to make use of the Makaton symbol, a standard technique, to help students master the basics of vocabulary by associating hand signs and symbols with objects. A number of simulations based on the Makaton symbol were developed and students could interact with the 3-D simulation while still keeping the 2-D Makaton symbol in view. By replacing the previous static printed information with an interactive simulation, students showed better knowledge retention [BUR94].

Procedural knowledge is difficult to obtain because it has to be obtained through repetition of the procedure in real life. Simulation can be very effective for training of procedural knowledge since the real life situation is replaced with a virtual situation. Other visual methods of training involve showing pictures and video clips with sound to show a student what to do. But we all know that it is one thing to be instructed how to drive a motor car, and another to actually drive it.

Military training simulators for planes, submarines, tanks and helicopters normally incorporate a cockpit identical to that used in reality. The cockpit is then mounted on a motion system that works with complex hydraulics. Real-time computer-generated images are displayed using the same field of view that is seen from a real cockpit. Military training also involves infantry training and missile training [AND96, MAT96, VIN95].

2.3.4.6 Medicine

The provision of health care has been changed in the recent years by the increased computer usage in medicine. Virtual reality can be used for several applications, such as anatomy trainers, surgery simulation, telesurgery (remote surgery) and rehabilitation.

VR techniques can be used very successful in teaching students human anatomy and pathology. The conventional way of teaching students human anatomy makes use of textbooks and cadavers for dissection. Using a HMD to view a 3-D virtual human model, students can work with a virtual cadaver [BUR94].

Surgical simulators are very powerful tools. Novice surgeons can train, surgical planning of complex procedures can be done and even be rehearsed before a real procedure. Novice surgeons training on cadavers cannot repeat given procedures if a mistake is made, since the body organs have been altered. According to Burdea ([BUR94]), "an internationally known expert in eye surgery has told the authors that 'it takes thousands of operations to become really proficient.' Who would like to be in the first hundreds of cases?" This is why surgical simulators are such powerful tools. They allow surgeons to learn by repetition the way aeroplane pilots do, without harming themselves, other people or animals.

The concept of telesurgery is that a surgeon can "operate" locally on a virtual patient model while his actions are transmitted via high-speed networks or satellite to a robotic assistant operating on a real, but distant patient. NASA was interested in telesurgery in order to perform surgery in outer space from earth. Under-developed countries could also benefit from telesurgery since it can improve their health care.

VR can also be used in rehabilitation processes to bring faster recovery and increased patient motivation. Various barriers have to be overcome in order to integrate into society. These barriers could be due to motor disability, which makes building access difficult, others relate to daily communication with people. One example that can help deaf people communicate with hearing people is the "talking glove". These are datagloves connected to a neural network gesture recogniser. The output from the neural network is then sent to a voice synthesiser for the hearing person. Therefore the hearing person does not need to understand sign/gesture language. The "GloveTalker" from Greenleaf Medical Systems is such a system [BUR94].

People with phobias can be treated using VR techniques. The Kaiser-Permanente Medical Group in Marin County, California, USA, has developed a trial system which evaluates the use of VR in the treatment of acrophobia, which is the fear of heights. 90% of participants reached self-assigned goals [VIN95]. A virtual aeroplane was developed in Georgia for the treatment of the fear of flying. Using this virtual aeroplane is much cheaper than exposure therapy. The subjects' self-reported anxiety decreased after a few sessions [HOD96].

Referring to the above examples, VR can be applied very successfully to the field of medicine. Some VR applications are used for training medical students, others for assisting medical personnel and others for helping and treating patients. More examples of medical applications are discussed in section 2.4. These are also more relevant to the system described in this thesis.

2.3.5 Advantages and Disadvantages

Advantages of virtual reality are easy to realise when applying it to dangerous environments and once-off scenarios. A trainee cannot get hurt or hurt someone else while practising dangerous tasks in a virtual world. Therefore VR is an excellent tool for training people to do dangerous tasks and lives may even be saved.

VR simulations can also save companies money because simulations can be used to detect faults in designs of products before production of the product. The world of entertainment benefits from VR since arcade and computer games seem much more real and attract more people. A virtual world can easily be manipulated, therefore it is easy to set up different scenarios. For example, practising to land a Boeing 747 in Japan under good weather conditions can easily be changed to bad weather conditions. The visualisation aspect of VR can be used very successfully with data visualisation. Experiments show that people trained via VR learn faster and make fewer mistakes than those who trained using traditional methods [HAM93, VRN96d].

Disadvantages of virtual reality applications are that they are still quite expensive to develop and that they require very powerful computers [BUR94]. With the rapid development of faster central processing units for personal computers, the cost of running virtual reality applications has definitely dropped to within reach of personal computers, but it is still expensive compared to other personal computer components.

Prolonged exposure to a virtual reality system can cause physiological disorders, like motion sickness, dizziness and ‘disorientation’. The reason for this is because vision is the most influential sense on the equilibrium and convincing generated visual data can cause

contradictory signals for the brain, which can cause nausea. It is believed that latency in image generation can also cause motion sickness [WIL95].

Focusing is very important with an HMD. If the optics are not focused properly it may lead to headaches due to the eyes straining in order to focus.

2.4 Previous and Related Work

This section will discuss work related to this study. Examples of existing virtual reality simulators will be discussed. It will also be explained how physical models created by artists and experiments on patients can be used for training. The evolution of medical training methods will be discussed, by first discussing experiments on patients, using physical models created by artists for training and then VR simulators.

2.4.1 Experiments on Patients

Medicine is one of the most ancient human occupations. In the 4th century BC, people like Hippocrates and Aristotle dissected many species and studied insect and animal behaviour with great accuracy. Aristotle believed that the scientific method of careful observation, experimentation, and study of cause and effect could lead to greater scientific knowledge. An 8-volume encyclopaedia on medicine known as “De Re Medica” was written by Celsus, a Roman that lived approximately between 10-37 AD. Two of these books treated topics in surgery, including operations for goiter, hernia, and bladder stone, as well as describing tonsillectomy and the removal of eye cataracts [GRO97]. Before anatomical models or computer simulators were available for medical students to practise on, medical students had to practise their skills by experimenting on patients. Even today, the first number of medical procedures that a student does under supervision of a specialist, can be seen as experimenting on live patients, because the student does not have any other means of gaining experience. In fact this is the conventional way of training medical students still today and therefore plays a big role in the medical training field.

2.4.2 Physical Models Created by Artists

Medical students can also be trained using physical life size models with photo realistic, artistically sculpted conditions (pathologies) inside. These models can be created by using real organs to make moulds. For the upper gastrointestinal region, a real stomach would be used to create such a mould. A non-transparent, plastic model of the stomach can be created, which can be painted on the inside like a normal stomach. Certain abnormal conditions, like ulcers and polyps, can be added inside the model. Therefore, using a real gastroscope (in working condition) to examine the inside of this model, the student would see the artistically sculpted conditions. Electrodes can also be added to certain places inside this model, for example at the site of the conditions, for practising therapeutic procedures like biopsy taking. As soon as the tip of a therapeutic tool touches one of the electrodes, an electric circuit will be closed to switch on a light, because the front tip of most of the therapeutic tools are of metal. This light will then indicate to the trainer where the student has touched the inside of the model.

The advantage of such a model is that it is very realistic because it is moulded from a real stomach. If a rubbery material were used to mould the model, the model would also feel very realistic. The disadvantage is of course that once a student has finished examining the model, he/she knows where the abnormal conditions are. In other words, the position of these conditions cannot be changed. This would mean that several different models should be built to accommodate a variety of conditions.

The system developed for this study also makes use of a physical model. The difference is that the physical model is transparent and the conditions are placed inside a virtual model that looks just like the physical model. This allows a trainee to see where the gastroscope is inside the physical model and conditions can be placed anywhere inside the virtual stomach. It is also not necessary to use a gastroscope that is in working order since only the positions and orientation of the gastroscope inside the physical model is needed for the system to generate a computer graphic image of how a real stomach would have looked like on the inside.

2.4.3 Virtual Reality Simulators

This section will first discuss some related endoscopic simulators, minimally invasive surgery simulators and then some other medical simulators.

2.4.3.1 GastroSim

Ixion's product, GastroSim, is an integrated multiprocessor system that provides real-time co-ordination of the gastrointestinal endoscopic experience for training purposes. The real-time manipulations are monitored by an expert system. The images used with the GastroSim are videos of the gastrointestinal tract and are said to be very realistic for simulating a natural organ turning, bending and flexing. By April 1996, the system had already been in beta test for several months at three teaching hospitals [VRN96a, VRN96b]. Recent searches revealed no information about this system.

2.4.3.2 Arthroscopic Simulator

Arthroscopy is a special endoscopical diagnosis method to recognise pathological changes and diseases of joints, for example knee, hip and shoulder.

A Virtual Arthroscopic Knee Surgery Simulator was developed at Sheffield University, U.K. Ligaments are currently modelled as simple cylinders, which are translated, rotated and scaled so that they always connect the two end points on the bones. A synthetic knee model is used with real instruments with electromagnetic trackers attached to the instruments. This simulator was developed on a Pentium 133MHz PC with a Matrox Millenium graphics accelerator card. An acceptable level of rendering speed is kept by using carefully designed objects to minimize polygon counts. In 1996 the cost of the system was around $38000 [HOL96, VRN96c].

An arthroscopic training simulator has been developed in Germany. A knee model was generated using magnetic resonance imaging (MRI) data. Acceptable performance has been achieved using a model consisting of nearly 20000 polygons. A real exploratory probe and a replica of an arthroscope are inserted into a synthetic model of the knee to make the interaction as realistic as possible. A frame rate between six and ten frames per second was achieved using a Silicon Graphics Indigo Extreme workstation [ZIE95].

2.4.3.3 Endoscopic Sinus Surgery Simulator

Endoscopic Sinus Surgery (ESS) is the treatment of medically resistant recurrent acute and chronic sinusitis. Basically it is a procedure to improve the natural drainage of the sinuses into the nasal airway. An ESS simulator has been developed at the Ohio State University, Columbus. This simulator consists of a mock patient head with nostrils. It uses a volumetric model of the anatomical region and delivers haptic feedback. A Silicon Graphics workstation ONYX/RE is used together with an endoscope and a five-degrees-of-freedom probe, the Microscribe™ by Immersion Corporatoin of San José. A frame rate of up to 20 Hz was obtained. One drawback that exists is that the image is degraded as the endoscope approaches objects for a closer look. This is caused by the lack of higher data acquisition in the volumetric model. In the future the system will incorporate volume deformation algorithms and colour by texture mapping actual photographic representations of nasal mucous membranes on the surface of the displayed data [YAG96].

2.4.3.4 Gynaecology Training Simulator

Ames-Iowa Engineering Animation, Inc. (EAI) has developed an endoscopic training simulator. Their Virtual Hysteroscopy simulator runs on a Silicon Graphics workstation with a model hysteroscope and a 3-D sensing device. Physicians can be taught how to diagnose uterus conditions and to perform surgical procedures on a 3-D animated model of the uterus. [VRN96b]

2.4.3.5 Minimally Invasive Surgery (MIS)

With minimally invasive surgery a surgeon inserts various tools through one or two small incisions in the chest, abdomen, spine or pelvis. The visual feedback is on a monitor or through an eyepiece [YAG96]. Major surgery can be done through these small incisions. Fast recovery is therefore possible because only a few stitches are required instead of a large incision through the skin and muscles which requires several stitches. MIS also ensures less pain, less need for post-surgical pain medication, less scarring and less likelihood for incisional complications. Virtual reality can be very useful in simulations for surgical training. MIS simulations involve inserting instruments through small openings and displaying a computer-generated model overlaid upon visuals of surgical representations. Some examples of existing systems follow.

Cine-Med is a medical education company who is developing a high cost VR skills simulator for MIS [VRN96a]. According to their web page (www.cine-med.com), their “Virtual Clinic” is a highly realistic, interactive training system which surgeons can use to gain clinical expertise. A study was conducted to find out approximately how many hours of practising on the simulator are required before a significant difference in performance could be seen in comparison to surgeons with no MIS training. Three surgical teaching hospitals were included in this study with a total of 36 subjects, of which 25 were experimental subjects and 11 were control subjects. The study concluded that surgeons would need about 45 hours of practice before a significant difference in performance was seen.

High Techsplanations’ (HT) surgical simulation system is also a high cost system. In 1996 the cost of their simulator was approximately $250000, but this problem was addressed by leasing the complete VR system for $3000 a month to medical colleges [VRN96a]. Recent searches revealed that their “PreOp Endovascular Simulator” is still under development. According to HT’s web page (www.ht.com), the “PreOp Endovascular Simulator” can be used to train clinicians for procedures such as balloon angioplasty and stent placement. Tactile feedback is used so that clinicians manipulating these devices can "feel" sensations experienced during procedures, such as encountering an unexpected obstruction in the artery.

Minimally Invasive Surgery Training by Virtual Reality (MISTVR), developed by Virtual Presence London U.K., is a virtual reality training simulator that can be used to train standard psycho-motor skills for laparascopic surgery. Virtual laparoscopic interfaces can also be added to the system, which will enable realistic force feedback. The system runs on a Pentium 133 MHz with Windows NT and in 1996, sold for about $15000 [VRN96b].

A training simulator for laparoscopic surgery in gynaecology has been developed in Lille, France. The inner cavity and organs have been modelled by hand using modelling software with the help of anatomical books, videos and measurements taken on real patients. No texture mapping is used. This simulator has been developed on a Pentium Pro 200 MHz with a GLZ5 OpenGL graphics board and the system runs on Windows NT [JAM97].

2.4.3.6 Ultrasound Training Simulator

The UltraSim, by MEDSIM, is an advanced ultrasound training simulator that allows students to practice sonographic exams on a mannequin while viewing real sonographic images. The simulator looks very realistic and feels very realistic. According to MEDSIM's web page (www.medsim.com), the mannequin provided with the system can also speak and answer questions like real patients.

2.4.3.7 Retinal Laser Photocoagulation Simulator

Retinal laser photocoagulation is a surgical technique used for the treatment of retinal diseases. A training simulator has been developed at the University for Science and Technologies of Lille, France. Modelling of the eye was done using spheres and half-spheres. The technique of photomapping real images of the eye onto the geometric objects was used to show realistic images of the eye. The simulator was implemented on a 80486DX2 66 MHz. The simulator's user interface is run on another PC that is linked to the simulator PC with a serial cable. A frame rate of about 8 frames per second was achieved in a 640x400 resolution using 32768 colours [MES95].

2.4.3.8 Virtual Reality Assisted Surgery Program

The Mayo Clinic has designed the Virtual Reality Assisted Surgery Program (VRASP) for eventual use during craniofacial, orhtopedic, prostate and neurologic surgery. VRASP lets doctors view 3-D renderings of CT and MRI data to plan and rehearse surgery so it will be more effective, less risky and less expensive [ROB96].

2.4.3.9 Green Telepresence Surgery System

The Green Telepresence Surgery System consists of two components, the surgical workstation and a remote worksite. At the remote site there is a 3-D camera system and responsive manipulators with sensory input. At the workstation is a 3-D monitor and dexterous handles with force feedback. The VR surgical simulator is a stylised recreation of the human abdomen with several essential organs. Using a HMD and glove, a person can learn anatomy or practice surgical procedures [SAT94].

2.4.4 Conventional Training Methods vs. Virtual Reality Simulators

The advantage of using medical simulators and physical models is that a student can gain experience by repeating certain procedures without harming the patient. The student can therefore become more confident with certain procedures. Because simulators and physical models could never replace the experience of doing a real procedure, the danger exists that the student could learn bad habits since he/she knows that it is not a real procedure. Therefore “experiments” on patients should never be omitted from training. Training simulators and physical models should rather be seen as tools that can help prepare a student for conventional training.

2.5 Summary

This chapter was used to give a literature overview of this study. The two main fields of study involved are gastrointestinal and endoscopic procedures, and virtual reality. Endoscopy refers to the examination of the internal organs, such as the stomach, gal bladder or duodenum, using a thin instrument inserted through body openings, like the mouth or rectum. Gastrointestinal endoscopy is a subspecialty of internal medicine that focuses on the intestinal tract.

The term VR refers to a system that generates real-time interactive visual, audio and/or haptic experiences. VR hardware and software was discussed in this chapter. VR display devices like HMDs and shutter glasses were discussed as well as different kinds of data gloves and their functionality. Several tracking devices, like ultrasound and electromagnetic devices, were discussed and the working of each was explained. The term VR software refers to software modelling tools for building virtual worlds, simulating physical behaviours in virtual worlds and rendering software for representing the virtual worlds. Software modelling tools are used to construct 3-D computer graphic models. Rendering software is used to convert the 3-D geometric descriptions of graphical objects into 2-D image plane representations that look real. VR is used in a number of fields, like entertainment, engineering, science, medicine and training. Work related to this study, like similar existing simulators, was also discussed.

Chapter 3

Overview of the Simulator's Main Data Sets and Components

3.1 Introduction

The simulator's main components and data sets will be introduced and a diagrammatic representation of them will be used as the structure according to which the following chapters will be presented. The next section will discuss the three main data sets and the last section will discuss the main components.

3.2 The Simulator's Main Data Sets

The simulator needs certain input data to be able to simulate what happens in the real world. With this simulator there are three main sets of input data, the VR model, the region database and the 3-D condition database. The VR model, which is the most important data set, is a computer graphic model of the stomach. The region database

contains information about certain anatomic regions inside the stomach, like the pylorus or duodenum. The 3-D condition database contains information about abnormal conditions that could occur inside the stomach, like ulcers or polyps. Figure 3.1 shows a diagrammatic representation of the three main data sets of the simulator. They all act as input for the VR system, which is the software application that the end user will use.

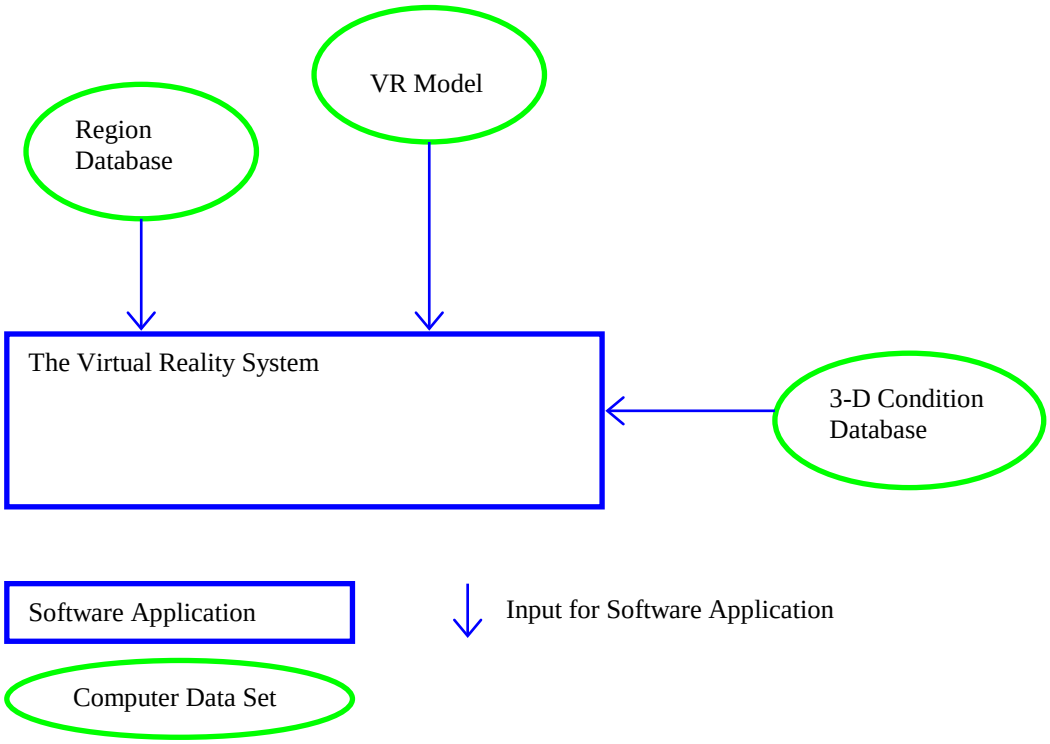

Figure 3.1 – Diagrammatic representation of the simulator's three main data sets.[1]

3.3 The Simulator's Main Components

The previous section briefly introduced the three main data sets of the simulator. This section will introduce the main components of the simulator after which it should be clear how the three main data sets were generated. Figure 3.2 shows a diagrammatic representation of the simulator's main components, as well as the processes involved. The following chapters will also make use of Figure 3.2 to show the reader exactly which data sets, processes and components are discussed in each chapter. The following sections in this chapter will introduce the reader to the components and processes shown in Figure 3.2.

[1] The meanings of the symbols used in this figure are explained at the bottom of the figure. For example, all green elliptic symbols are computer data sets.

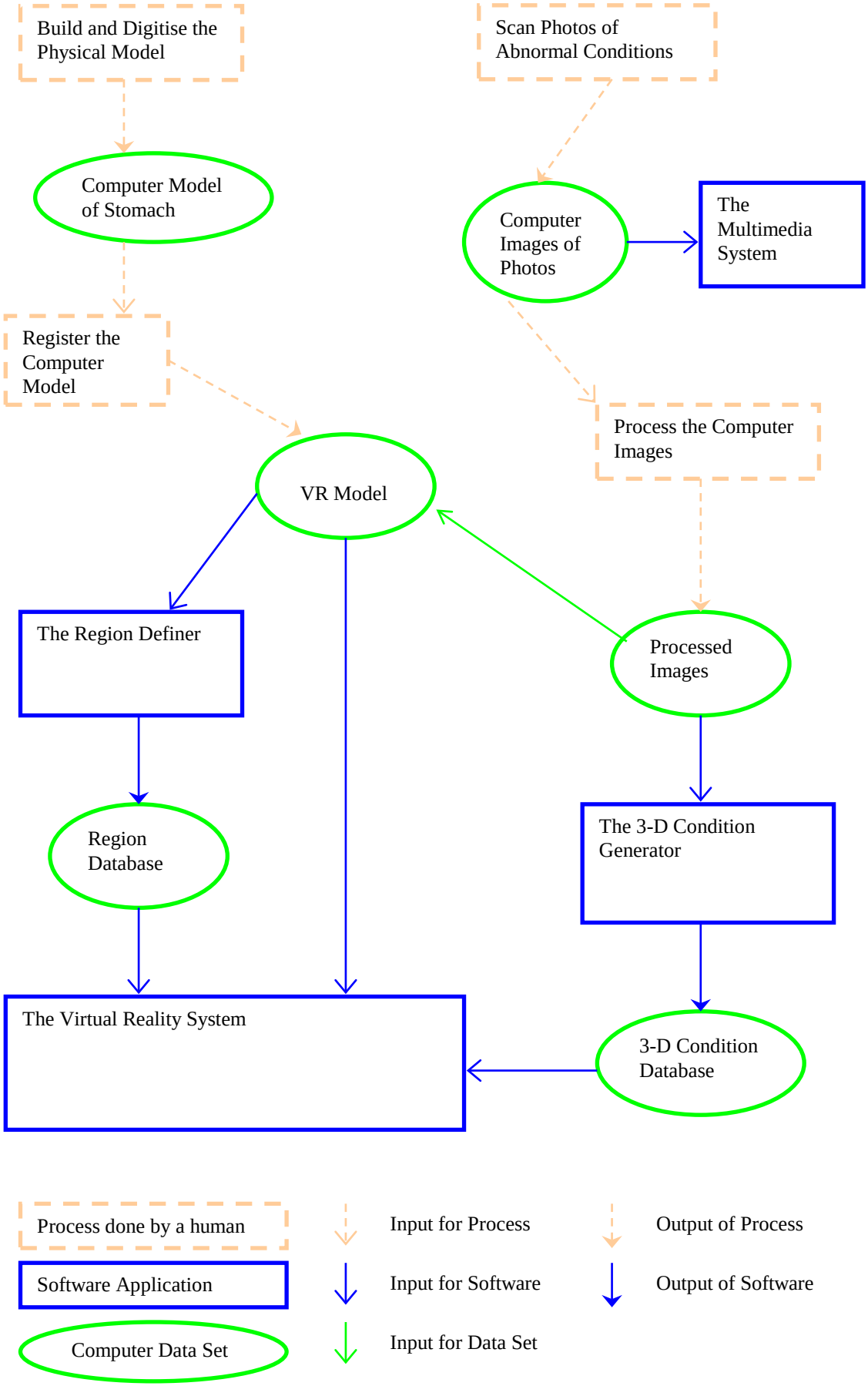

Figure 3.2 – Diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system.

3.3.1 Build and Digitise the Physical Model

Referring to Figure 3.2, *build and digitise the physical model* is a human process of which the output is a computer data set that is referred to as the *computer model of stomach*.

A physical model of the stomach had to be built, which is the model that the trainees will use to practise on. This model is hollow and transparent so that the trainees can see what happens on the inside of the stomach while practising. The inside shape of the stomach was digitised (scanned) to get a computer graphic model of the stomach which shape is the same as that of the inside of the physical model. At that stage the computer graphic model's scaling and orientation was not correlated with that of the physical model, so it was just called the computer model. Chapter 4 will discuss the digitising of the physical model in more detail.

3.3.2 Register the Computer Model

Register the computer model is a human process that uses the *computer model*, which is a computer data set, as input to deliver the (registered) *VR model*.

This process is very important, because if the computer model is not registered correctly, the simulator will not function correctly [ALI97]. Registration involves the calibration of scale, position and orientation of the computer model so that it correlates with the physical model. The closer the VR model's scale, orientation and position can be correlated to that of the physical model, the more realistic the simulation will be. If the computer model is not registered, then a person can, for example place the tip of the gastroscope in the physical model's esophagus, but in the VR model, the tip will be shown at the pylorus sphincter. Chapter 4 will discuss the registration of the computer model in detail.

3.3.3 Scan Photos of Abnormal Conditions

This is a human process that results in the output of a computer data set, which is called *computer images of photos*.

One of the most important factors that will make a computer graphic simulation realistic is what the user sees. To simulate the inside of a stomach, an artist could generate images that look real, or a colour scanner could be used to scan photos taken by a real gastroscope and use these photos in the simulator.

This simulator makes use of real images, which enhances the simulation. These images can also be used in a multimedia system for diagnostic training. Refer to Chapter 6 for more detail on this process.

3.3.4 The Multimedia System

The multimedia system is a software application that uses the *computer images of photos* to display case studies. Chapter 6 has more detail on this software application.

3.3.5 Process the Computer Images

Process the computer images is a human process that uses the *computer images of photos* as input, to deliver the *processed images*. Processing of the images is necessary as the images cannot be used in the exact form they were scanned. Each condition needs to be processed so that it can be "fitted" smoothly into the stomach. This image processing is done with standard image processing software. Chapter 6 has more detail on this process.

3.3.6 The Region Definer

The region definer is a software application that uses the *VR model* as input to deliver the *region database*.

The region definer is used to define certain anatomic regions of the stomach or any organ for that matter. This data can be saved as a region database and is used for training of navigational skills. Chapter 5 has a detailed discussion about the region definer.

3.3.7 The 3-D Condition Generator

The 3-D condition generator is a software application that uses the *processed images* as input. The output is the *3-D condition database*.

The condition generator is a very important tool used to generate 3-D abnormal conditions that can be placed anywhere inside the VR model of the stomach. The generator uses the processed images of real abnormal conditions and certain mathematical functions to generate 3-D conditions. Chapter 6 has a detailed discussion about the 3-D condition generator.

3.3.8 The Virtual Reality System

The virtual reality system is a software application that uses *the VR model*, *the region database* and *the 3-D condition database* as input to simulate gastrointestinal procedures. Chapter 7 will discuss this software application in detail.

3.4 Summary

This chapter gave an overview of all the main data sets, components and processes of the simulator and broadly describes how they are involved with each other. The following chapters will describe in detail each of the three main computer data sets, as well as the other components and processes involved. Chapter 4 will discuss the VR model, Chapter 5 the region database, Chapter 6 the 3-D condition database and Chapter 7 the virtual reality system.

Chapter 4

The VR Model

4.1 Introduction

The VR model is a computer graphic model of the stomach. This model is used to simulate what a real stomach would look like on the inside. To accomplish this, the VR model should have the same 3-D shape as a real stomach and should have the same colour, texture and orientation as a real stomach inside a human body. Pictures of real stomach tissue are used to ensure that the simulation looks as realistic as possible.

Figure 4.1 shows a diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system. The red block shows the overall process of how the VR model was generated. It also shows which data sets and processes are involved with the generation of the VR model. This chapter will discuss these processes so that the reader can understand the process of generating the VR model.

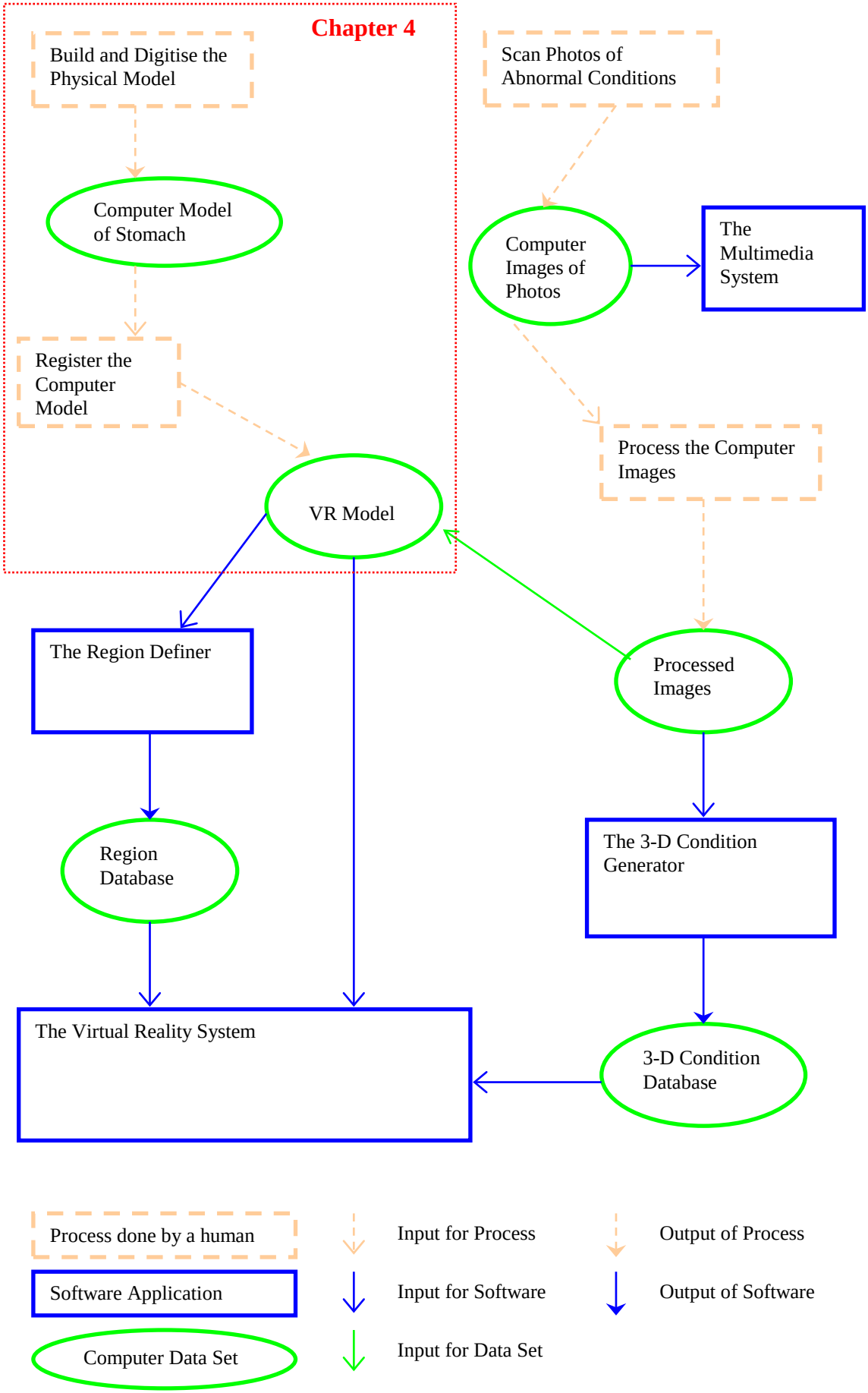

Figure 4.1 - Diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system.

4.2 Build and Digitise the Physical Model

A physical model of the stomach had to be built. The trainees will practise on it by inserting a gastroscope into the model. The model is hollow and transparent so that the trainees can see what happens on the inside of the stomach while practising. To build a hollow, transparent, life size model of the stomach, the following was done.

First a solid model had to be moulded that has the same shape and size as a real stomach. This solid model represents the shape of the stomach on the inside. Refer to the movie clip "**\Movie Clips\3- Showing the Physical Model**".[2] Using a 3-D tracking device and digitising software, this model was digitised (scanned) according to the predefined design specifications, which will be discussed in the following section. This basically implied that certain points on the physical model had to be marked and measured according to a fixed 3-D axis. These measured points are called vertices and can be used to create polygons. Refer to the next section for more detail on computer graphic models. This digitised data was then saved as a text file so that it could be imported into any 3-D editor. The digitised data will be referred to as the computer model.

The hollow transparent model still had to be built. This is the actual model into which the trainees will place the gastroscope to practise. Using the solid model, two moulds, one for the front and one for the back were created to build the hollow transparent model. To make the model transparent, fibreglass was used. This model will be referred to as the physical model.

The end result was two pieces of fibreglass that could be fitted together to form a hollow transparent stomach. Transparent Perspex pipes were added to the openings of the stomach model to represent the esophagus and duodenum. Plate 4.6 shows the physical model from the side. The two pieces fitted together can be seen in this plate.

[2] The building of the physical model and digitising was done by 5DT <Fifth Dimension Technologies>

4.2.1 Design of the VR Model

Setting up design specifications for the VR model is important, because it specifies how the VR model must be created while digitising it. Having design specifications for the VR model will simplify the process of creating other models, like a gall bladder, for the simulator.

Before the design specifications will be discussed, some knowledge of how computer graphic models are created, is necessary. Section 4.2.1.1 will discuss the creation of 3-D computer graphic models, section 4.2.1.2 will discuss texture mapping and section 4.2.1.3 will discuss the design specifications of the VR model.

4.2.1.1 Computer Graphic Models

Computer graphic models are constructed using a set of 3-D vertices and a set of polygons. The 3-D vertices are the 3-D points that give the model a specific 3-D shape. These vertices can be obtained using 3-D digitising equipment. If these vertices are connected, polygons are formed. Each polygon therefore represents a surface that has a set of connected vertices. See Figure 4.2 and Figure 4.3. Also refer to Plate 4.1 and Plate 4.2.

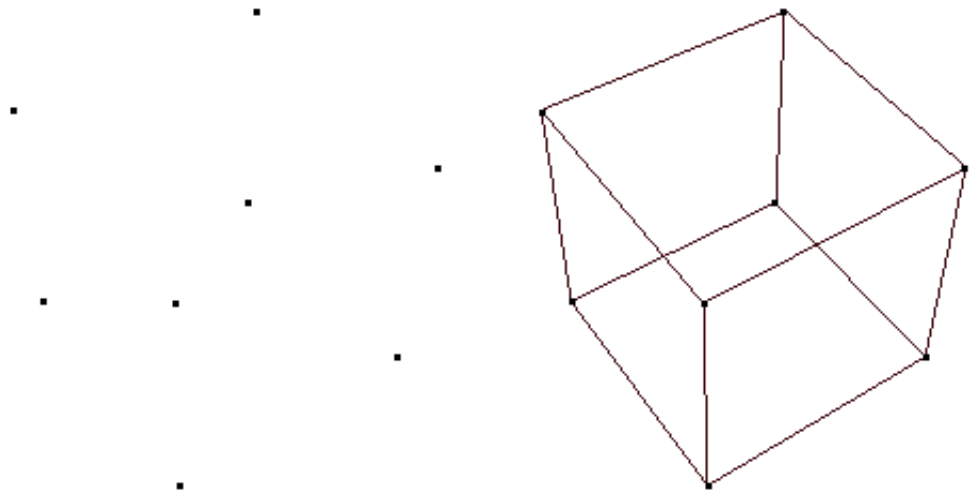

Figure 4.2 - A cube’s vertices on the left side and its polygons on the right side.

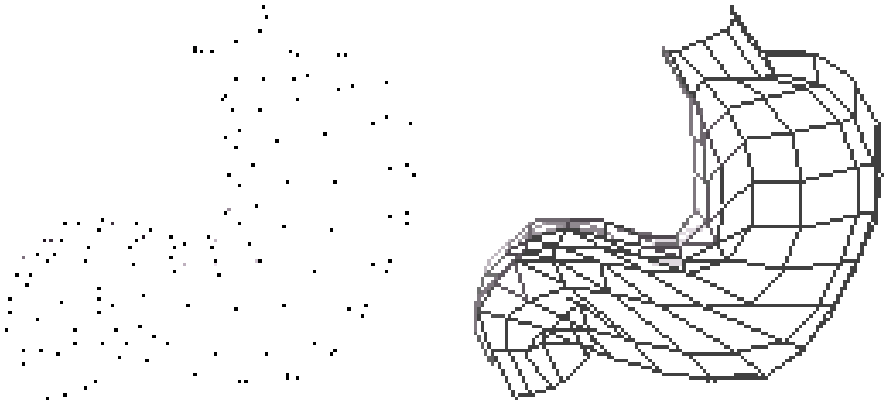

Figure 4.3 - A model of a 3-D stomach's vertices on the left side and its polygons on the right side.

Each polygon has two faces or surfaces, a front face and a back face. Obviously only one can be seen at a time depending on the viewpoint. For example, referring to Figure 4.2, if the viewpoint is inside the cube, the surfaces facing the inside of each polygon will be seen. If the viewpoint is outside the cube, the surfaces facing the outside of each polygon will be seen. To optimise rendering, only one of the faces is normally used. To specify which face has to be used, a direction vector has to be calculated for each polygon. This direction vector is called the normal vector of the polygon. The normal vector can be calculated as follows:

Two vectors must be calculated by subtracting the vertices of the specific polygon from one another. For example, if a polygon has four vertices, then the first vector can be calculated by subtracting vertex two from vertex one, and the second vector can be calculated by subtracting vertex three from vertex two. The first vector is in the same direction from vertex one to vertex two and the second vector is in the same direction from vertex two to vertex three. See Figure 4.4.

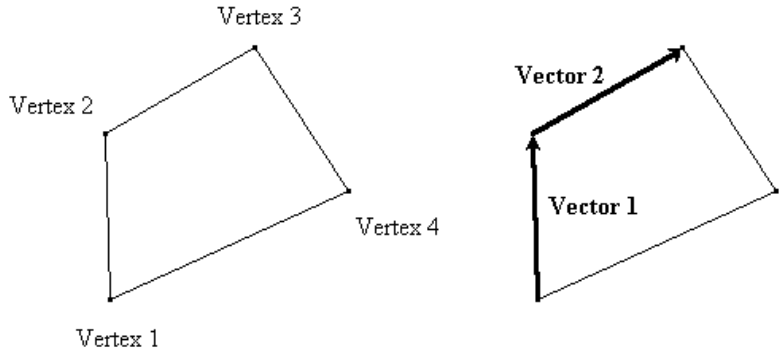

Figure 4.4 – A polygon with four vertices and the two vectors that must be calculated to determine the direction of the normal vector of the polygon.

Calculating the cross product between vector one and vector two results in the normal vector of the polygon. The polygon's normal vector can easily be changed by reversing vector one and two. This can be used to decide whether the inside or outside of a 3-D object has to be shown. For example, this simulator's main window shows the inside of the stomach, but the same model has to be used to show the outside shape of the stomach in another window. Therefore, to show the outside shape, the polygons' normal vectors need to be reversed. See Figure 4.5.

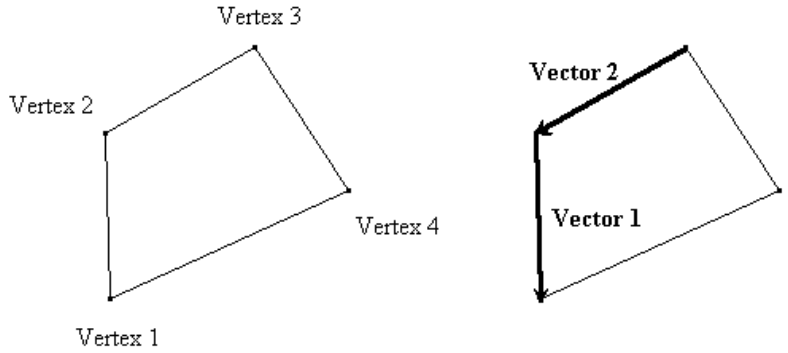

Figure 4.5 – A polygon with four vertices and two vectors that must be calculated. These vectors are calculated in the opposite direction as the ones in Figure 4.4 so that the normal vector of the polygon can be reversed.

A polygon can either be planar or non-planar. If it is a planar polygon then all its vertices reside on a single Cartesian plane. Triangles are always planar polygons and that is why rendering software works with triangles. Sometimes it is easier to work with polygons that

have more than three vertices. In this case, these polygons must first be triangulated before it can be rendered. Normally the rendering software does the triangulation during initialisation. For more information on triangulation of polygons, refer to Programming Principles in Computer Graphics by Leendart Ammeraal [AMM92].

Using more vertices and polygons to create the model will result in a model appearing more continuous or smooth. Unfortunately it will also result in slower rendering since more calculations will have to be done for transforming all the vertices and rendering all the polygons. Refer to Chapter 7, section 7.4.1.2 for more detail on this topic.

Polygons can be assigned certain colours, which could be sufficient in some cases, like simulating a red ball, but very often much more detail is needed on a 3-D model. For example, if a simple house has to be modelled, with a roof and four walls, the walls can be brown and the roof black. But it would look much more realistic if somehow we can "paste" a picture of roof tiles onto the polygons that represent the roof and a picture of bricks onto the polygons that represent the walls. The next section will discuss how to add such finer detail to a 3-D model.

4.2.1.2 Texture Mapping

According to F.M. Weinhaus and V. Devarajan ([WEI97]), the term texture mapping refers to "the technique of integrating photographs of real-world scenes with real-time computer-generated images to enhance realism." A texture normally represents a computer image of a real-world texture, like a brick, bark of a tree, or metal. A texture can be stretched over a polygon. Using textures, very fine detail can be added to the computer graphic model. See Chapter 6 for detailed information on how textures were prepared so that it could be used with the VR model of this system. When large viewing ranges have to be accommodated, for example, the surface can be viewed from far away or from close by, a group of texture maps can be used to represent one image. These are called mip maps and will ensure that a high quality image resolution is maintained for all the viewing ranges.

Colours are very important when working with images. Working in a display environment of 256 colours or less, indexed colour palettes have to be used to accommodate all the colours in a texture (keep in mind that this system was developed for the IBM PC). An indexed colour palette is basically an array of colours. For example, an indexed colour palette with 256 colours will be an array with 256 positions. Each of these positions can represent a colour by using some colour model, like the Young-Helmholtz model where each colour consists of three additive primary components: red, green and blue [VIN95]. The advantage of using indexed colour palettes is that it works very fast in terms of processing speed. The disadvantage is that high-resolution textures will loose a lot of detail. Refer to Plate 4.3 for an example of a 256 colour image.

Working in a display environment of 16-bit high colour or higher, no indexed colour palettes are needed. The advantage is that high resolution textures will not loose any detail when rendered. The disadvantage is that it is slower than working with indexed colour palettes. See Plate 4.4 for an example of the latter. Refer to Chapter 9 for a detailed discussion of the testing of this system using colour palettes versus no colour palettes.

Now that the reader is more familiar with 3-D computer graphic models, the next section will discuss the design specifications of the VR model.

4.2.1.3 Design Specifications of the VR Model

Before a physical model can be digitised (scanned), some specifications have to be set up as to how the computer graphic model's vertices and polygons must be constructed. This will make it easier to digitise other models to add to the system, because it will be done in the same way. This section will describe this simulator's 3-D stomach design specifications.

Firstly, each polygon in the computer model should have its own texture. The reason for this is to maintain a good image resolution. Renderware version 2.1, the render engine (being used in this project), specifies that textures should be of size 256x256 or 128x128. It was found that if only a few textures of these sizes are used, the level of detail is very low and the image gets very blocky. But if each polygon has its own texture, the level of

detail is very high. This will also ensure a high level a detail when the tip of the gastroscope is very close to a polygon.

Secondly, all the polygons in the VR model must be quad-shaped polygons. Because textures are rectangular and each polygon will have its own texture, it is much easier to work with quad-shaped polygons than for example triangular polygons. Renderware will automatically triangulate each quad-shaped polygon so that it can be rendered.

Thirdly, the polygons should be arranged as a rectangular grid of quad-shaped polygons so that there is always a polygon to the left, right, top and bottom of any given polygon, except for the polygons at the end of the duodenum and esophagus. The reason for this is to be able to tile textures smoothly so that it looks like the inside of a stomach. The best way to explain this is if one would take a rectangular grid of quad-shaped polygons and wrap it around the solid mould of the inside of the physical model. See Figure 4.6.

Figure 4.6 – The basic design specifications for the VR model.

Lastly, each polygon in the model must be tagged with a positive integer. These polygon tags will be used when defining anatomic regions. See Chapter 5 for more detail on anatomic regions.

4.3 Register the Computer Model

Registration of the computer model is very important, because if the VR model is not registered correctly, the simulator will not function correctly [ALI97]. Registration of the computer model involves the calibration of scale, position and orientation of the computer model so that it correlates with the physical model. The closer the VR model's scale,

orientation and position can be correlated to that of the physical model, the more realistic the simulation will be. If the VR model is not registered, then one could for example place the tip of the gastroscope in the physical model's esophagus, but in the VR model, the tip will be placed at the pylorus sphincter.

To understand why it is important to register the computer model, it must be stressed that the simulator uses two models of the stomach. One is a physical model in which the trainees place a real gastroscope and the other is a computer graphic model which is used to show the trainees what the stomach looks like on the inside.

To simplify things, it was decided to digitise the model so that if a person lies on his/her left side, the esophagus will be along the x-axis. Keeping this in mind, the most correct anatomical orientation was used to digitise the model. The physical model was then built and mounted inside a Perspex case, also with its orientation as anatomically correct as possible. Unfortunately there will always be a difference in orientation between these two models and because the orientation of the physical model is fixed in the Perspex case, the computer model's orientation has to be changed so that it corresponds to that of the physical model. Further, the VR model is scaled and translated to unit space when the virtual reality system starts because it is easier to do calculations in unit space. Because of this scaling and translation, a *scaling factor* has to be calculated to apply to the 3-D tracking device's input, so that the correct viewpoint from the tip of the gastroscope can be shown.

The following sections will discuss two methods used for the registration of the computer model. The first method is an analytical approach and the second one is an empirical approach. Because these methods involve 3-D transformations, it was thought best to first give a broad overview of how 3-D transformation matrices can be used to transform 3-D points.

4.3.1 3-D Transformation Matrices

Geometrical transformations can be used to size, rotate and express the location of an object relative to another object. Any 3-D point can be transformed into another point using a 4x4 matrix. Matrices are written in row-column order. Note that all transformation matrices explained in this section are for a left-handed coordinate system. Equation (4-1) shows how point (x, y, z) is transformed to a new point (x', y', z').

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix} \tag{4-1}$$

To get the new point (x', y', z'), the following operations must be done on (x, y, z) and the matrix:

$$x' = (x \times M_{11}) + (y \times M_{21}) + (z \times M_{31}) + (1 \times M_{41}) \tag{4-2}$$

$$y' = (x \times M_{12}) + (y \times M_{22}) + (z \times M_{32}) + (1 \times M_{42}) \tag{4-3}$$

$$z' = (x \times M_{13}) + (y \times M_{23}) + (z \times M_{33}) + (1 \times M_{43}) \tag{4-4}$$

Transformations that can be done are scaling, translating and rotating. Examples of these three transformations will be shown and discussed in the following paragraphs.

Equation (4-5) shows a transformation that scales the point (x, y, z) by arbitrary values in the x-, y-, and z-directions to a new point (x', y', z') so that the original point's x-coordinate is multiplied by S_x, the y-coordinate multiplied by S_y and the z-coordinate multiplied by S_z.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-5}$$

A matrix that evenly scales vertices along each axis (known as uniform scaling) would be represented by a matrix where S_x and S_y and Sz are equal to each other. Equation (4-6) shows such a matrix.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-6}$$

Equation (4-7) shows a transformation that translates the point (x, y, z) to a new point (x', y', z') so that the original point's x-coordinate is moved along the x-axis by distance T_x, its y-coordinate is moved along the y-axis by distance T_y and its z-coordinate is moved along the z-axis by distance Tz.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \tag{4-7}$$

Equation (4-8) shows a transformation that rotates the point (x, y, z) around the x-axis by the angle α, producing a new point (x', y', z').

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-8}$$

Equation (4-9) shows a transformation that rotates the point (x, y, z) around the y-axis by the angle α, producing a new point (x', y', z').

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-9}$$

Equation (4-10) shows a transformation that rotates the point (x, y, z) around the z-axis by the angle α, producing a new point (x', y', z').

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4\text{-}10)$$

Matrices can be combined to calculate several transformations at once. This is called matrix concatenation. Matrix concatenation is obtained by multiplying the separate matrices in a certain order. This means that, to rotate a model and then translate it to some location, only one matrix needs to be applied and not two. Instead, the rotation and translation matrices are multiplied to produce a composite matrix that contains the whole of their effects. This process can be written with equation (4-11).

$$C = M_n \cdot M_{n-1} \ldots M_2 \cdot M_1 \quad (4\text{-}11)$$

In equation (4-11) C is the composite matrix being created, and M_1 through M_n are the individual transformations matrices that matrix C will contain. Notice the order in which the matrix multiplication is performed. The preceding equation reflects the right-to-left rule of matrix concatenation. This means that the visible effects of the matrices used to create a composite matrix occur in right-to-left order. For example, creating the transformation matrix for a rolling ball requires a rotation transformation and a translation transformation. To obtain the effect of the ball rotating around its own centre and moving in some direction, equation (4-12) can be used.

$$W = T_w R_z \quad (4\text{-}12)$$

In equation (4-12) T_w is a translation matrix to some position in world coordinates and R_z is a matrix for rotation about the z-axis. The order in which the matrices are applied is important because, unlike multiplying two scalar values, matrix multiplication is not commutative. Multiplying the translation and rotation matrices in the opposite order would

have the visual effect of translating the ball to its new position, then rotating it around the *world* origin and not the *ball*'s origin.

4.3.2 Methods of Registration

As was mentioned already, there are three things that need to be calculated to register the computer model. A scaling factor for scaling the computer model, a translation matrix for moving the computer model to the correct position and a rotation matrix for orientating the computer model to correlate with the physical model. The scaling and translation are not that difficult, since it only involves measuring the physical model's width, height and offset position from the origin of the 3-D tracking device. Figure 4.7, Plate 4.5 and Plate 4.6 illustrate how the measuring was done. Note on Plates 4.5 and 4.6 where the 3-D tracking device's transmitter is. This is taken as the origin of the 3-D tracking device.

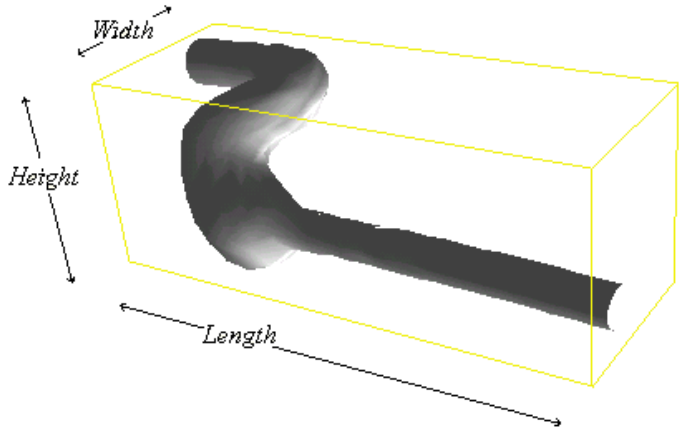

Figure 4.7 – Shows how the physical model was measured.

What is left to do is to rotate the computer model so that its orientation is the same as the physical model's orientation. This is the most difficult part, because the orientation involves rotation around the x-, y- and z-axes. The duodenum and esophagus can be used to simplify this task. The duodenum and esophagus can be seen as being parallel to the x-axis. This means that the rotation around the y- and z-axes can easily be done. Any 3-D editor, like 3-D Studio, can be used to align the esophagus and duodenum parallel to the x-axis. Figure 4.8 illustrates how the esophagus and duodenum is aligned parallel to the x-axis. Therefore only the rotation around the x-axis needs to be *calculated*.

The ideal would be to digitise the physical model only after it has been mounted in the Perspex case with its orientation fixed. This is unfortunately impossible since the whole of the inside of the physical model has to be digitised. Two methods have been used to register the computer model. Both will be described below. For both methods, the first thing to do, was to mount the physical model with a fixed orientation, so that it was only necessary to rotate the computer model to correlate with the physical model, and not the other way round.

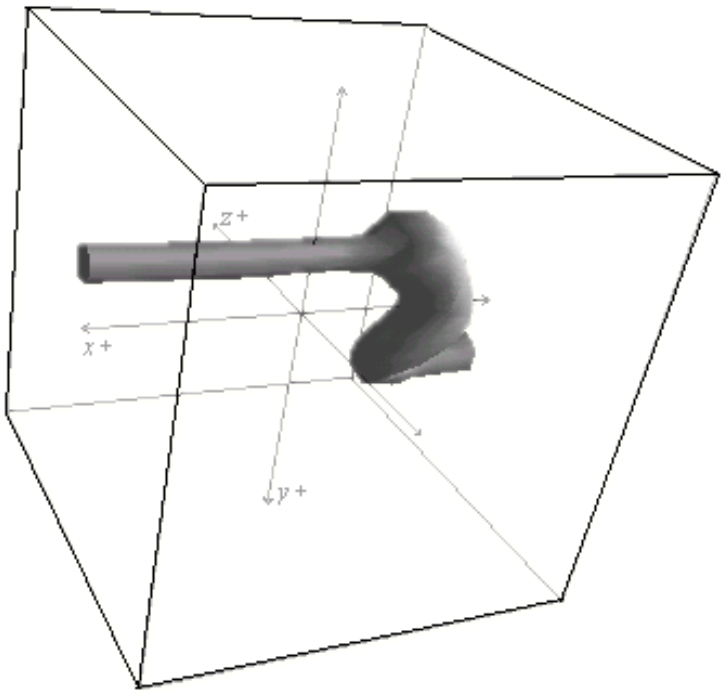

Figure 4.8 – The duodenum and esophagus parallel to the x-axis.

4.3.2.1 Method 1 – The Analytical Approach

The first method used was to let the computer automatically calculate the correct rotation around the x-axis of the computer model. To do this, the fact that the number calculated by dividing the fixed physical model's height by it's width should be exactly the same for the computer model when it is in the same orientation, even if the two models are scaled differently, were used. This resulted in equation (4-13).

$$\frac{PhysicalModel(Height)}{PhysicalModel(Width)} = \frac{ComputerModel(Height)}{ComputerModel(Width)} = Const \quad (4\text{-}13)$$

Theoretically speaking the *Length* of the models could also be used to calculate a constant, but unfortunately the physical model's length is not fixed since it depends on how far the Perspex pipes (representing the esophagus) fit into the fibreglass.

The physical model was mounted inside a Perspex case and its width and height were measured, and the *Const* factor could be calculated. A small software application was developed that uses the physical model's *Const* factor to find the optimum rotation around the x-axis so that the computer model's *Const* factor is about the same as that of the physical model's. The application basically worked on the same principles as the binary search algorithm works on. It starts with a lower boundary rotation, which is 0 degrees, and an upper boundary rotation, which is 180 degrees. A test value is calculated, using:

$$\frac{UpperBoundary - LowerBoundary}{2} = TestRotation \quad (4\text{-}14)$$

The test value is used to rotate the computer model around the x-axis. The computer model's height and width is then calculated for the new orientation, the *Const* factor is calculated and tested to see how close it is to the physical model's *Const* factor. If it is not close enough, the domain between the upper and lower boundaries are separated into two smaller domains by setting the upper and lower boundaries, using:

First domain: $LowerBoundary = TestRotation$ (4-15)

Second domain: $UpperBoundary = TestRotation$ (4-16)

The same procedure is then done on both smaller domains. The one which rotation results in giving the closer *Const* factor will be used further, until a rotation is found that results in giving a *Const* factor close enough to that of the physical model's.

The first thing that became apparent with the implemention of the above was that there were always more than one solution to the problem. This means that more than one possible rotation around the x-axis will result in a *Const* factor for the computer model that is almost the same as that of the physical model's. The second thing that was noticed was that neither of these rotations resulted in an orientation for the computer model that looked almost the same as the physical model's orientation. One of the reasons for not getting good results might be that the rotations around the y- and z-axes could not have been correlated close enough with that of the physical model, using a 3-D editor. Another reason could be of parallax errors that occurred during the physical measurements inside the Perspex case. This method of finding the best orientation was not chosen.

4.3.2.2 Method 2 – The Empirical Approach

The second method used was to rotate the computer model around the x-axis until it *looked* close enough to the orientation of the physical model. This method is called a free hand method, since it did not involve any automatic calculations. As mentioned in the previous method, only the rotation around the x-axis had to be found, because the duodenum and esophagus of the physical model are parallel to the x-axis. Because both of them are parallel to the x-axis, they are actually parallel to one another. This means that by just looking carefully at the physical model, the computer model could also be rotated around the x-axis so that the computer model's duodenum and esophagus are lined up in the same way that the physical model's duodenum and esophagus are lined up. The result of this method was a VR model that had almost the same orientation as the mounted physical model. This method of finding the best orientation was chosen.

During the registration of the computer model, the actual data or 3-D vertices were scaled, translated and rotated. This means that after the computer model has been registered, it can be used as the VR model in the system and the system does not have to register the computer model each time it starts. The system does scale and translate the VR model to unit space when it starts, but this is only done because it is easier to work in unit space when maximum and minimum values have to be calculated.

4.4 Conclusion

The purpose of this chapter was to explain what the VR model is and how it is generated by first digitising a physical model to get a computer model and then registering the computer model to get the VR model. The main problem discussed in this chapter was how to get a computer graphic model that has the same shape, size and orientation as a normal stomach. The VR model that was generated proved to be very successful and the system can render it quite fast. The design specifications were adequate, but since only one VR model (the stomach) has been generated for this system, it cannot be known if the design specifications are flexible enough for generating VR models of other organs. The registration of the VR model also proved to work well using the empirical approach.

Chapter 5

The Region Database

5.1 Introduction

In this chapter it will be discussed what the region database is, why it is needed and how it is generated. It will also be discussed how the region definer is used, what its main components are and how it is implemented.

Regions refer to the anatomic regions of the (gastrointestinal) organs defined in the medical literature, for example the esophagus, duodenum or pylorus sphincter. These anatomic regions' names can vary, but normally they do not vary much. Regions are used in this system to indicate certain areas inside the (gastrointestinal) organs for orientation purposes. The region database is a database that has information of these regions and is mainly used during navigational training to orientate and to test a trainee.

Figure 5.1 shows a diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system. The red block shows the overall process of how the region database is generated. It also shows which data set and component are involved with the generation of the region database. This chapter will discuss this component so that the reader can understand the process of

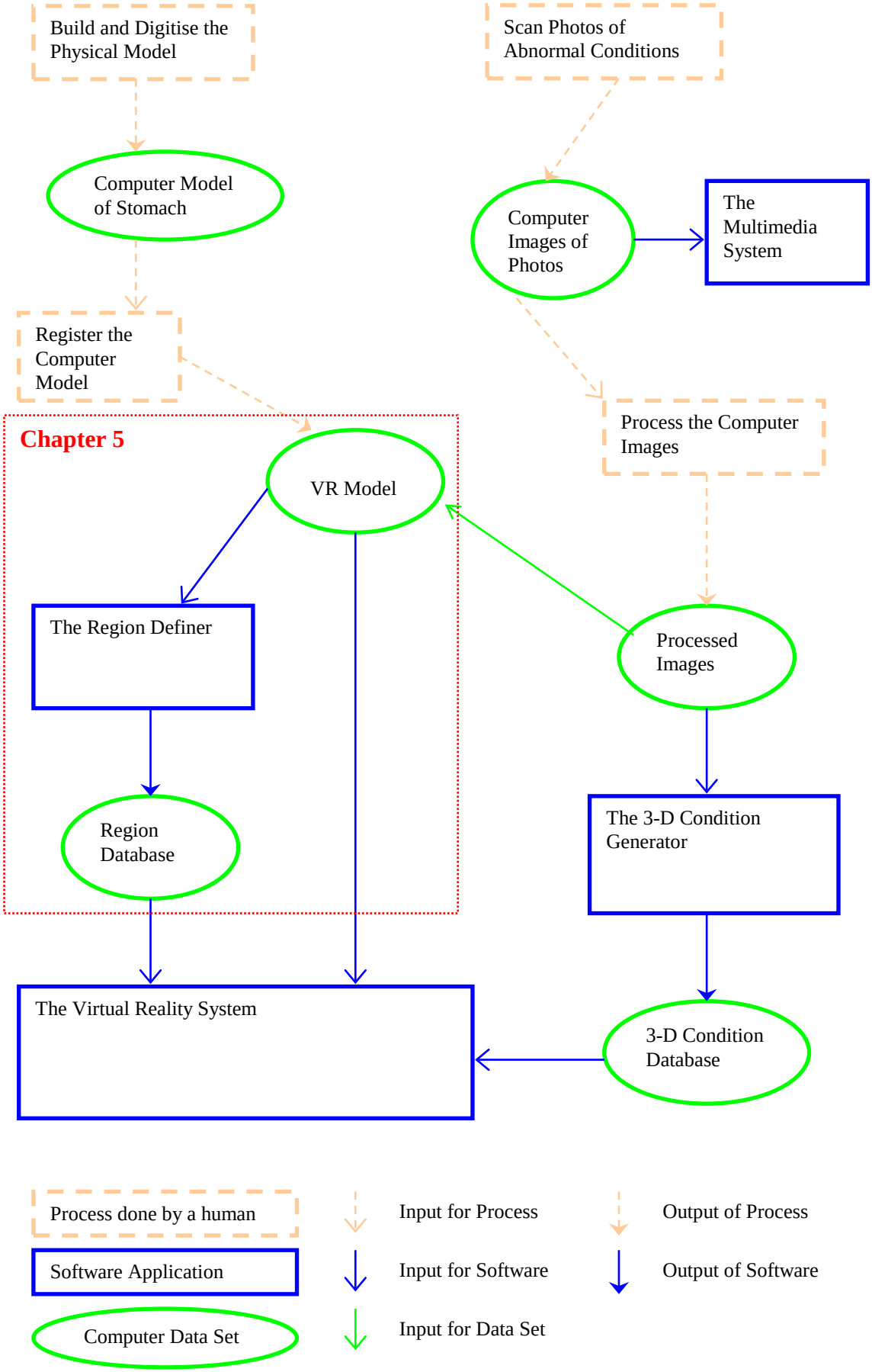

Figure 5.1 - Diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system.

generating the region database. Note that the VR model data set is also included in the red block since the VR model is used as input for the region definer.

5.2 The Region Definer

The region definer is a software application developed to define certain regions of the VR model of the stomach. These regions can be saved in a region database and is used during the training of navigational skills in the virtual reality system.

In the following sections it will be discussed how the region definer can be used to define the anatomical regions. The main components of the region definer, how they were implemented and the data sets used by the region definer, will also be discussed.

5.2.1 Using the Region Definer

Firstly a VR model which was digitised and registered must be loaded, in this case the VR model of the stomach. A 3-D model of the stomach will then be shown on the screen, which can be rotated in any direction, using the left mouse button. The regions must then be defined by adding a region name to the region name list and defining which polygons are part of that region. To define which polygons are part of a region, the right mouse button can be used to click on a polygon to be added to the current region. The colour of the polygon will change from light grey to red. All the red polygons are part of the current region. See Plate 5.1. When all the regions have been defined, the region database can be saved for further use in the virtual reality system.

The menu was implemented in the following way:

File

Load Organ

Loads a VR model which regions have to be defined.

Load Regions

Loads an existing region database.

Save Regions

Saves the current region database.

Exit

Exits from the software application.

Edit

Undo

When this menu item is marked, clicking with the right mouse button will clear the current polygon and remove it from the current region.

Clear All Marked Polygons

Clears all the marked polygons of the current region.

View

Reverse Polygon Faces

Reverses all the polygons' normal vectors so that the VR model can be viewed from the inside or from the outside. This is useful since the VR model is generated to be viewed from the inside, but it is easier to define the regions working with the VR model viewing it from the outside.

Wireframe

When this item is checked, all the polygons in the VR model will be shown as wireframe, otherwise it will be solid. This could be useful to see how the VR model's polygons are defined to form the shape of the VR model.

Help

Help

Shows the help file of this software application. This help file has not yet been completed.

About

Shows information about this software application.

Refer to the movie clip "**\Movie Clips\Region Definer\Region Definer.avi**" on the accompanied CD-ROM for a demonstration of how this application works.

5.2.2 Overview of the Region Definer's Main Components

Figure 5.2 shows a diagrammatic representation of the main components of the region definer and the data sets used by the region definer. The following sections will discuss these components and data sets.

5.2.2.1 Initialisation

Referring to Figure 5.2, the *initialisation* module initialises this software application by firstly creating the *computer graphic user interface,* using standard Windows 95 API functions which should also be compatible with Windows '98. A *region database* with no records is created and initialised. The *render engine*, in this case Renderware, has to be initialised before any of its functions can be used. A virtual world, which is called a scene, has to be created. The *VR model*, which regions have to be defined, will be placed in this virtual world using the *computer graphic user interface*. A virtual camera has to be created and initialised to show the *VR model* in the scene. A light source has to be created to show light reflecting off the model in the scene, otherwise the camera image will just be dark (not illuminated). Refer to Chapter 2, section 2.3.3.2 for more detail on rendering software.

5.2.2.2 Computer Graphic User Interface

The *computer graphic user interface* basically consists of a main window with a menu. The mouse and keyboard can be used as physical input devices to use the menu and to create new or edit existing *region database* files. The particular *VR model* for which regions have to be defined, has to be loaded and initialised using the menu. See Plate 5.1. The region definer (and all other applications involved with this system) was developed

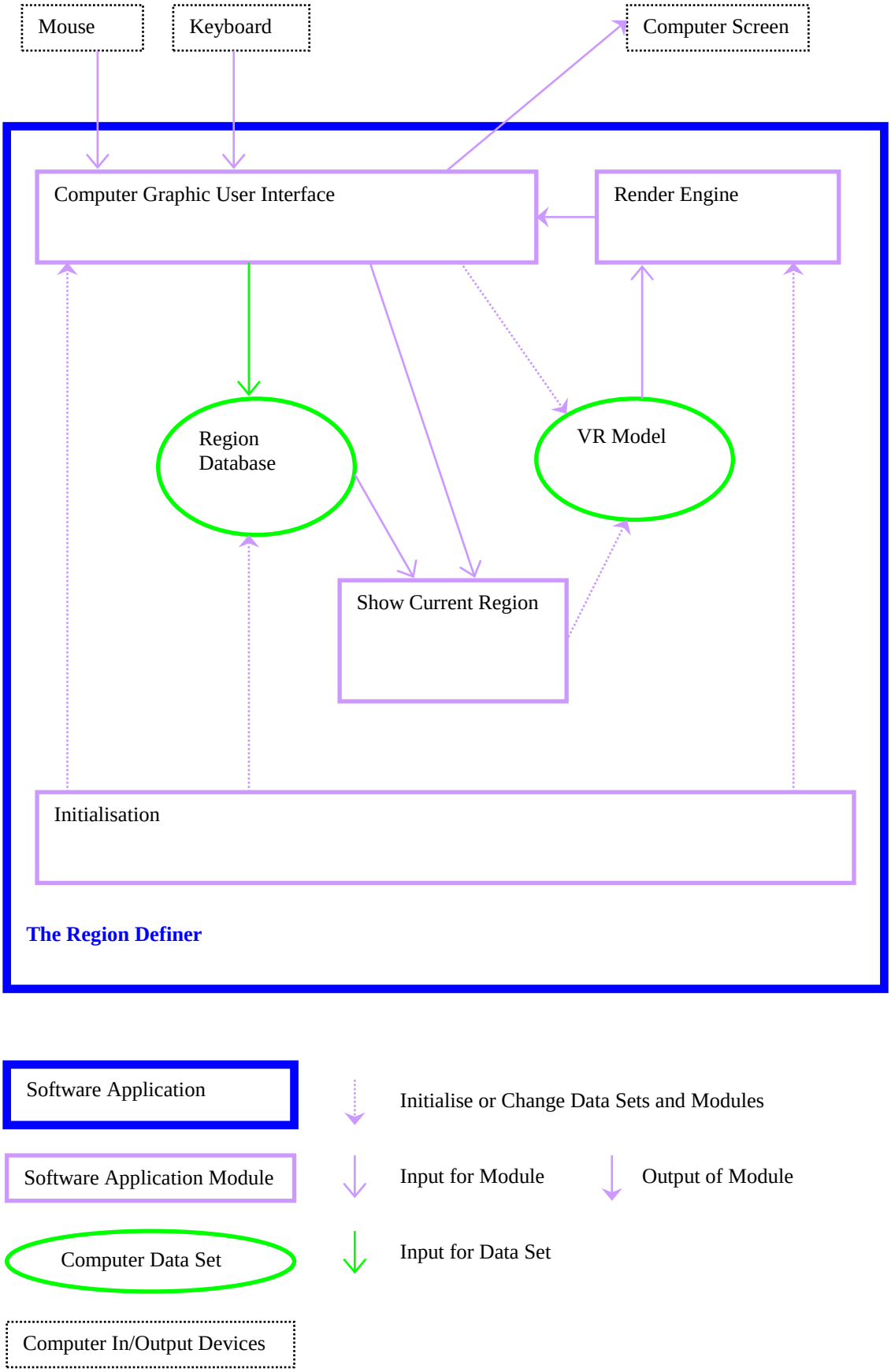

Figure 5.2 – Diagrammatic representation of the region definer's components and data sets used by the region definer.

using Watcom C/C++ version 10.6. Therefore all the routines, including the computer graphic user interface, were developed using C++ code with Windows 95 API functions.

5.2.2.3 VR Model

The *VR model* is a computer graphic model of the stomach. It is placed into the scene that was created by the *initialisation* module. The *VR model* serves as input for the *render engine* to show the stomach on the screen. Refer to Chapter 4 for more detail on the *VR model*.

5.2.2.4 Region Database

The *VR model* contains information about the shape and texture of the virtual stomach model while the *region database* contains information about the anatomical regions. Each polygon in the VR model may only be part of one region.

Each record in the database has the following structure:
Region Name
Polygon List

Region Name is a string that represents the name of the region and *Polygon List* is an array of integers that represents a list of polygons that are part of the region.

For example, the data record:

Pylorus
24 119 105 91 77 62 47 32 16 0 222

means that a region with the name “Pylorus” exists and polygons 24, 119, 105, etc. are part of the “Pylorus” region.

Since no complex queries will ever be done on the database, the database was implemented using a text file. In future it might be considered to implement the database as a “real” database, like a paradox database.

5.2.2.5 Show Current Region

This module is used to show the polygons of the current region by changing their colour. As soon as the current region is changed from the *computer graphic user interface*, this module changes the colour of all the currently marked polygons to the original colour and changes the colour of all those that are part of the newly selected region. If a polygon is added or removed from the current region, this module also shows the polygon in the correct colour. This module uses the *region database* as input to determine which polygons belong to which regions. The output of this module is a changed *VR model*, in the sense that only the colours of some of the *VR model*'s polygons' have changed. These changes are not permanent and are used only to show which polygons are in the currently selected region.

5.2.2.6 Render Engine

The *render engine* is the software module that takes as input all the 3-D objects in a 3-D scene, and calculates the 2-D image from the 3-D camera's viewpoint, making use of the positions of the camera and light source(s) in the scene. This module's output is relayed to the *computer graphic user interface* to be displayed on the screen.

The render engine used for the region definer (and all other applications of this system) is Renderware version 2 from Criterion. The reason for choosing Renderware, although it is quite expensive, is because of its rendering speed and quality and the fact that its rendering API functions are very easy to use. When this project was started in 1996, Renderware was the best choice. If the same project were to be started today, it is most likely that Microsoft's DirectX version 6 would be used since it is free and the rendering speed and quality are also very good. DirectX's API functions are not as easy to use as Renderware, but it is the author's point of view that this is due to poor help files.

5.3 Summary

The purpose of this chapter was to explain what the region database is and how it is generated. The region database has information about anatomical regions of organs used with the virtual reality system, like the stomach. The region definer is used to generate the

region database. The region definer's working and implementation was discussed as well as the structure of the region database. Also in this chapter is a short discussion on the render engine used for the region definer and all the other applications of this system.

Chapter 6

The 3-D Condition Database

6.1 Introduction

This chapter will explain what the 3-D condition database is, why it is needed and how 3-D conditions are created using photos of abnormal conditions. The author developed new methods for *generating* 3-D conditions, making the 3-D conditions *look realistic*, and for *blending* 3-D conditions into the VR model.

Basic identification training skills can be taught by placing abnormal conditions inside the VR model, which a trainee then has to identify. At first it was thought that placing flat 2-D pictures of abnormal conditions inside the VR model would suffice. This method involved the following: To set up a scenario, the instructor would choose a condition from a database of 2-D pictures and click on any polygon inside the VR model. The selected polygon's texture would then be set to the 2-D picture. If a condition had to be shown larger, a tiling method could be used [PAT97]. These larger conditions could be called complex conditions, because more than one image would be used to create one condition. Each complex condition would be prepared by resizing the original image and cutting the image into tiles of 256x256. Each tile would be a separate image that could be used to place inside the VR model on a single polygon. Plate 6.1 shows a photo of an abnormal condition and Plate 6.2 shows how the photo was resized and cut into four separate images for the tiling method. If the instructor would choose to add a complex condition to the VR model, he/she would click on a polygon inside the VR model where the upper left corner of the complex condition should appear. For the tile images to be laid out correctly and form the original image, the VR model's polygons must form a grid, because given any polygon in the VR model, it must be known which polygon is to another polygon's left, right, top and bottom. Each tile image is then applied to the appropriate polygon. Therefore the visible size of each condition is determined by the surface of the polygons over which it is applied. Refer to section 4.2.1.3 for the design specifications of the VR model. This method worked well for "flat" conditions, like bleeding. The advantage of this method is that it is easy and fast to implement, since only the texture of a polygon or polygons have to be changed. The rendering speed is also not affected, since no extra vertices or polygons need to be added to the VR model. The disadvantage is that conditions like polyps, that have a 3-D structure, do not look very realistic.

Most of the abnormal conditions are not flat structures. A good example of this is a polyp. Looking at Plate 6.1 a flat 2-D picture of a polyp can be seen. A normal human being should not have any problems interpreting the flat 2-D picture as a 3-D structure. The problem with the flat 2-D picture is that it has fixed shadows and that the 3-D structure cannot be viewed from a different angle. If it were to be placed inside the VR model, then the user must be able to view the condition from any direction. Therefore 2-D pictures of abnormal conditions cannot always be used realistically inside the VR model. A new

method was developed to overcome this problem. This method uses generated 3-D structures with 2-D images applied onto the 3-D structures to simulate abnormal conditions. This method will be discussed in detail in this chapter.

A 3-D condition is therefore a computer-generated representation of an abnormal condition, for example ulcers, polyps or gastritis, that can occur inside the gastrointestinal organs. The 3-D condition database is a database that has information about these 3-D conditions. One of the purposes of this training simulator is to teach a trainee basic identification skills. While training with these 3-D conditions, a trainee can learn to identify the basic abnormal conditions. Note that diagnostic skills are taught using photos of real conditions, rather than using computer-generated images. See section 6.3 for more detail on training diagnostic skills.

Figure 6.1 shows a diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system. The red block shows the overall process of how the 3-D condition database is generated. It also shows which processes, data sets and components are involved with the generation of the 3-D condition database. This chapter will discuss these processes and components so that the reader can understand the process of generating the 3-D condition database.

The 3-D condition database is created by first scanning photos of real abnormal conditions and then processing them. Using the 3-D condition generator together with these images, certain 3-D conditions can be generated. Information about these 3-D conditions together with references to the files containing the actual 3-D information are then added to a database file. The following sections will discuss this process of generating 3-D conditions in detail.

6.2 Scan Photos of Abnormal Conditions

For the system to be as realistic as possible, photos of real abnormal conditions are used. Dr. David M. Martin of Atlanta South Gastroenterology Private Clinic made his endoscopic atlas of gastric diseases available for this project. The atlas was obtained in the form of photos. These photos were scanned at 300 dpi (dots per inch) using a colour

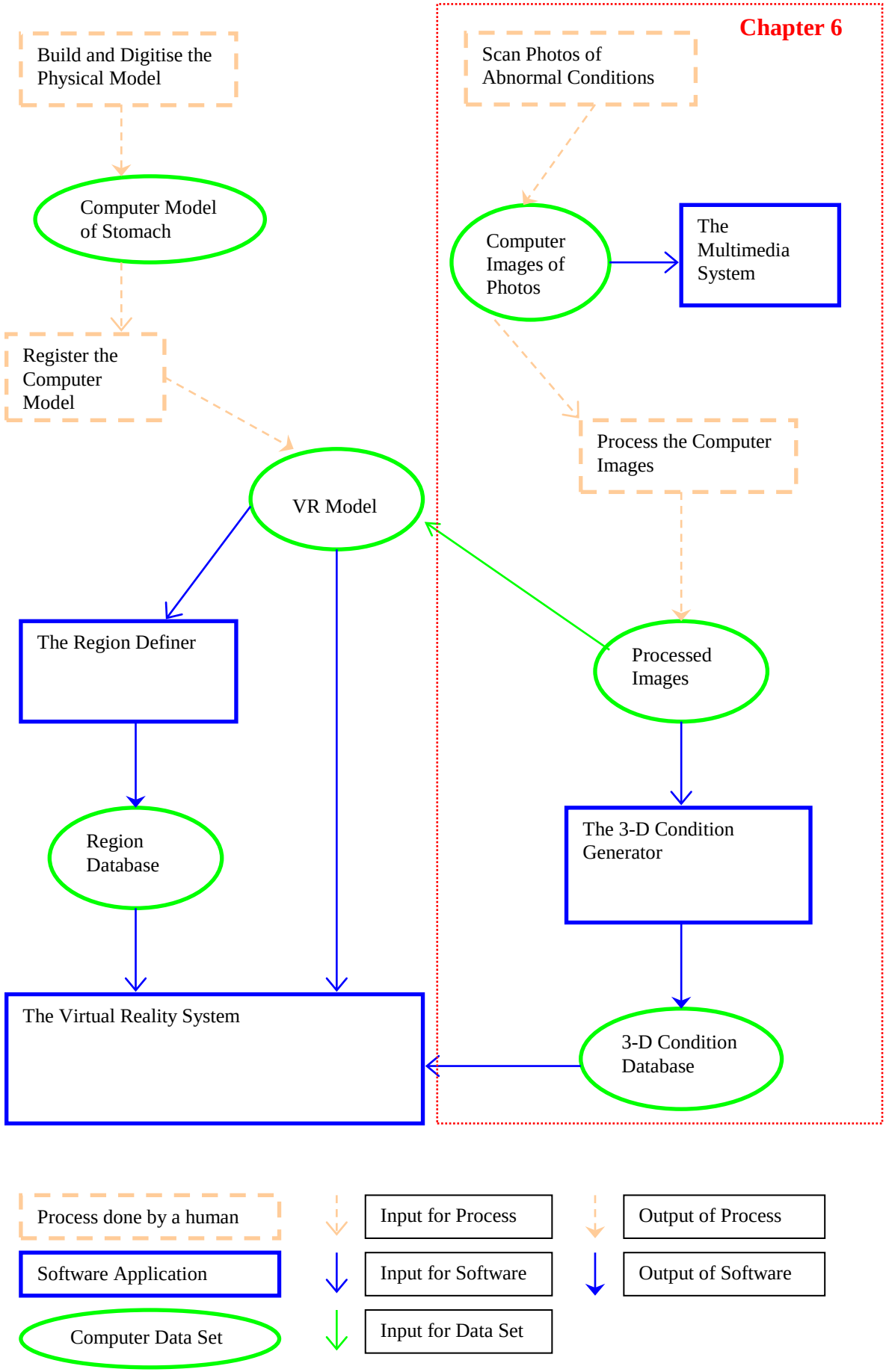

Figure 6.1 - Diagrammatic representation of the system's components and data sets, and processes and data sets involved in the development of the system.

scanner and saved in the standard Windows bitmap (.bmp) file format. The images were then scaled to 512x512 size images[3].

6.3 The Multimedia System

The multimedia system was developed by 5DT <Fifth Dimension Technologies>, therefore it will not be discussed in detail. The computer images of real abnormal conditions were used to create an endoscopic atlas in the form of a multimedia program. This multimedia system focuses on teaching a trainee diagnostic skills rather than basic identification and navigational skills as with the VR system. Therefore a much wider range of conditions and variants of the same condition is used in this system. The use of multimedia has proven to be a very good teaching tool and by using this medical atlas with multimedia techniques, trainees should learn much more, much faster. The multimedia system could also be used by doctors to confirm conditions that occur very seldom. See Plate 6.1 for an example of how these images can be used, with a detailed description of the condition, as a computerised endoscopic atlas.

6.4 Process the Computer Images

Once the photos have been scanned, a number of computer images are available. These images have to be prepared or processed before they can be used in the system. The reason for this follows: The simulator uses a single, very uniform image to represent normal tissue. This image was chosen very carefully after comparing many different photos of normal stomach tissue. The normal tissue image can be tiled to represent large areas of normal tissue, therefore the texture of each polygon in the VR model was set to the normal tissue image. In Chapter 4 it was discussed why each polygon in the VR model should have its own texture.

If an image of an abnormal condition is placed amongst the images of the normal tissue, it will stand out very clearly since the borders of the new image will not tile or match the borders of that of its neighbours. A trainee will then easily see where the abnormal conditions were placed. To prevent this, each image of an abnormal condition has to be

[3] Scanning and image scaling done by 5DT <Fifth Dimension Technologies>.

processed so that it will blend into a grid of tiled images of normal tissue. Comparing Plate 6.6 and Plate 6.7 with one another, it is very clear that the abnormal condition added to the VR model in Plate 6.6 does not blend into the background and the condition in Plate 6.7 does blend into the background.

The easiest way to do this blending is to use the image of the normal tissue as a background image, and blend the image of the abnormal condition onto the background image. This will ensure that the new image can also be tiled next to images of normal tissue. See Plates 6.3, 6.4 and 6.5 for an example of how this could be done. Currently the image processing is done by hand, using Aldus PhotoStyler 2.0, but it could later be done automatically by the system. See Plate 6.7 for the end result. Also refer to the movie clip "**\Movie Clips\VR System\2- 16Bit High Colour.avi**" that illustrates two abnormal conditions inside the VR model.

6.5 The 3-D Condition Generator

The 3-D condition generator is a software application that is a very important tool used to generate 3-D abnormal conditions that can be placed anywhere inside the stomach. A user can use the 3-D condition generator to generate a flat 3-D grid and apply a processed image of a real abnormal condition onto the 3-D grid. When saved, this 3-D grid can be used as a 3-D condition in the 3-D condition database. The 3-D condition generator also allows the user to use certain mathematical functions to form the 3-D condition's shape.

The following sections will describe how to use the 3-D condition generator to create 3-D conditions that can be used in the virtual reality system. The design and implementation of the 3-D generator will also be discussed.

6.5.1 Using the 3-D Condition Generator

A 3-D condition can be generated by generating a rectangular grid of polygons and applying an image of a real condition onto the grid of polygons. To generate a new 3-D abnormal condition, use the File menu. The user will be prompted for the size of the new grid. Any uneven value between 3 and 19 may be entered. Currently the system does not

prevent the user from entering an even number, but rather adds the value one to the even number to make it an uneven number. In future this will be changed so that only uneven numbers can be entered. A value of 5 will generate a flat (5-1) by (5-1) grid of polygons in the XZ-plane. See Figure 6.2.

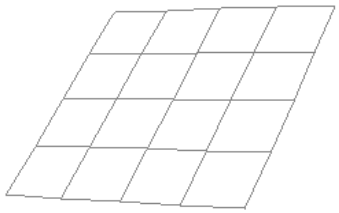

Figure 6.2 – A flat grid of 4x4 polygons generated. Note that it is a grid of 4 rows and 4 columns of polygons, but 5 rows of vertices and 5 columns of vertices are used.

The reason why the number of vertices must be an uneven number is to ensure that the grid has a vertex in the centre. If no vertex were in the centre of the grid, a symmetric shape would not always look smooth in the centre since the centre of the shape cannot be clearly defined. The grid can be rotated by holding down the left mouse button and dragging it either horizontally or vertically. This is useful if the user wants to view the 3-D condition from different angles. A processed computer image can now be loaded onto the grid. Plate 6.8 and the movie clip "**\Movie Clips\3-D Condition Generator\1- Flat 3-D Condition.avi**" illustrate how a processed computer image can be loaded onto the flat 3-D grid. To give the 3-D condition a 3-D shape, the user can edit and change the grid of polygons by applying certain predefined mathematical functions to the vertices of the polygons. Refer to Plate 6.9 and the movie clip "**\Movie Clips\3-D Condition Generator\2- Editing a 3-D Condition.avi**". After the 3-D condition has been generated, it can be saved.

The menu system was implemented in the following way:

File

New

Generates a new 3-D condition.

Load

Loads an existing 3-D condition.

Save

Saves the current 3-D condition.

Load Bitmap

Loads a computer image and maps it onto the current 3-D condition.

Exit

Exits the software application.

Edit

Shows a dialog box with which the user can choose a certain mathematical function *f(x)* and set its parameters. The function *f(x)* will be applied to the grid of polygons. Currently the user can choose between two functions, the Butterworth function or circle. Because the function can be scaled and distorted, the result of using the circle mostly looks like an ellipse function. The word “ellipse” is used in the drop down list that the user sees, and not “circle” because the user might associate the word “circle” with a perfectly round structure. A scroll bar identified by the word “Dent” can be used to scale the selected function either inwards or outwards along the y-axis. The first “Radius” scroll bar is used to set the value of the selected function’s radius, for example $y^2 = radius^2 - x^2$. The “Shift” scroll bar can be used to shift the function along the x-axis to get a mirroring effect. The “Gradient” scroll bar only effects the Butterworth function and can be used to make the function look “sharper” or “fatter”. The “Distort” scroll bar can be used to randomly distort the function for a more organic look. The last scroll bar, also “Radius”, can be used to set the radius where the distortion should be applied to the function. Refer to the movie clip “**\Movie Clips\3-D Condition Generator\2- Editing a 3-D Condition.avi**” for an example of how the edit functions can be used.

View

Wireframe

When checked, the condition is shown as a wireframe model. This is very useful to see what the 3-D structure and shape of the condition looks like.

Help

Help

Shows the software application's help file. This help file has not yet been completed.

About

Shows information about this software application.

Refer to the movie clips "**\Movie Clips\3-D Condition Generator\3- Generating an Ulcer.avi**" and "**Movie Clips\3-D Condition Generator\4- Generating a Polyp.avi**" to see how 3-D conditions can be generated using the 3-D condition generator.

6.5.2 Overview of the 3-D Condition Generator's Main Components

Figure 6.3 shows a diagrammatic representation of the main components of the 3-D condition generator and the data sets used by the 3-D condition generator. The following sections will discuss these components and data sets.

6.5.2.1 Initialisation

The *initialisation* module initialises this software application by firstly creating the *computer graphic user interface*. The *render engine*, in this case Renderware, has to be initialised before any of its functions can be used. A scene has to be created. A virtual camera has to be created and initialised to show the *3-D condition* in the scene. A light source has to be created to show light reflecting from the objects in the scene.

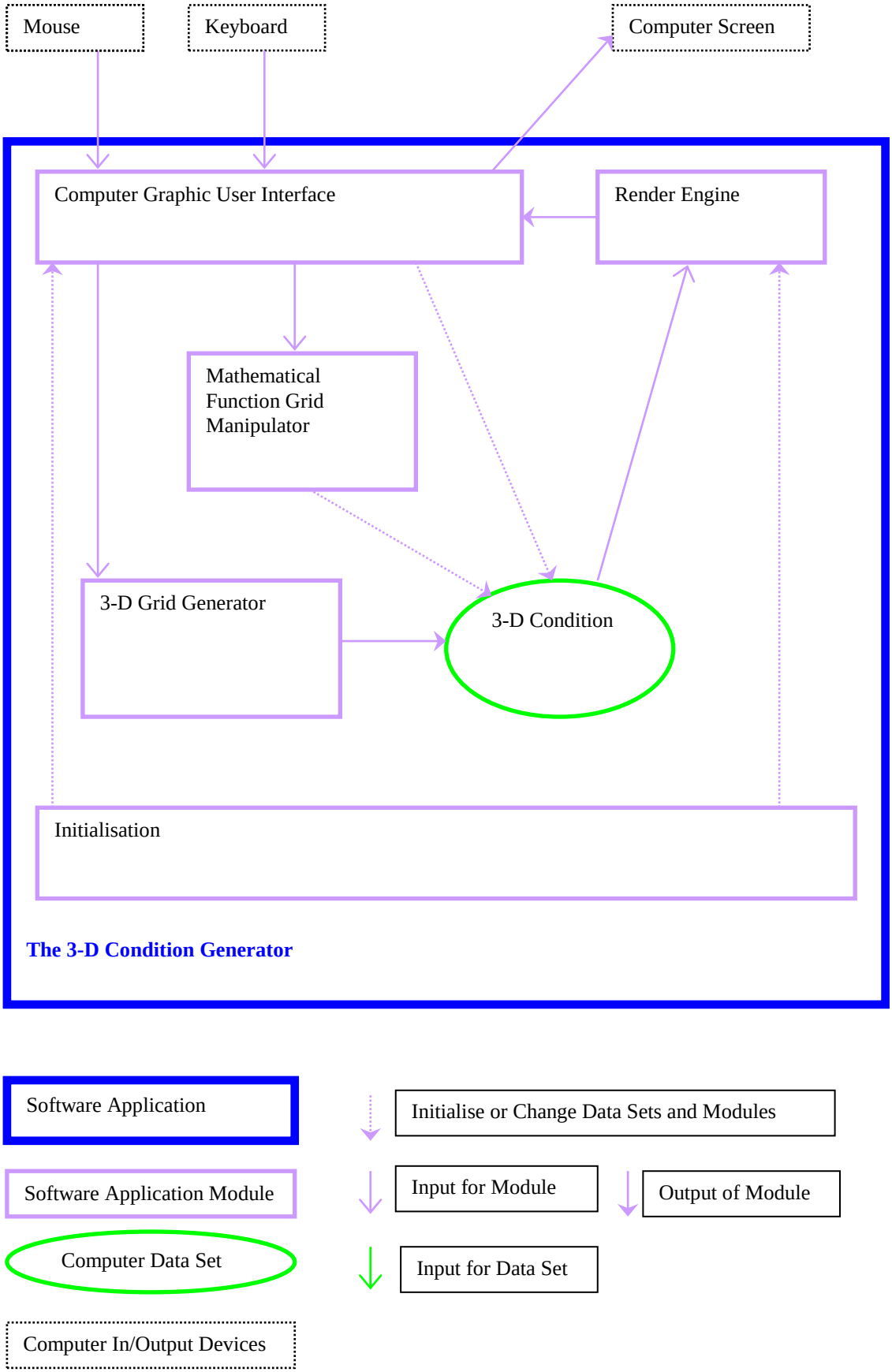

Figure 6.3 – Diagrammatic representation of the 3-D condition generator's main components and data sets used by the 3-D condition generator.

6.5.2.2 Computer Graphic User Interface

The *computer graphic user interface* basically consists of a main window, to display the 3-D condition, with a menu. A dialog box for selecting a mathematical function and setting its parameters is used to change the form of the condition. The mouse and keyboard can be used as physical input devices to use the menu and to generate new, or load existing *3-D condition* files. After a 3-D condition has been loaded or a new one created, the user can apply a processed image or change the current processed image that is shown on the 3-D condition using the menu. Only .bmp files can be used, since Renderware only allows for .bmp files to be loaded as textures. The 3-D condition generator was developed using Watcom C/C++ version 10.6. Therefore all the routines, including the computer graphic user interface, were developed using C++ code with Windows 95 API functions.

6.5.2.3 3-D Grid Generator

This module generates a flat rectangular 3-D grid of polygons in the XZ-plane, which is the base for any *3-D condition*. Only the *computer graphic user interface* interacts with this module since the 3-D grid generator is only activated when the user uses the menu when he/she wants to generate a new 3-D condition. Therefore the *computer graphic user interface* gives input to the *3-D grid generator* module. The output of this module is the *3-D condition*. The following paragraphs will discuss how the 3-D grid generator was implemented.

All 3-D conditions are generated in the XZ-plane. The user specifies the size of the base grid in terms of the number of rows or columns of vertices in the grid. The grid generated will be a square, or a x by x grid. A centre point for the grid is wanted, so the grid size must be an unequal number, for example 13. When the user enters the size, the vertices are generated from x = -1.0 to 1.0 and z = -1.0 to 1.0. These values are convenient to work with since it will generate the grid in unit space and the centre of the grid is at the origin (0,0,0). This simplifies calculations, like scaling. Note that the grid size has nothing to do with the actual size or surface of the grid. The grid size only refers to the number of rows or columns of vertices used to generate the grid. After the vertices have been generated, quad-shaped polygons are defined to form a flat square grid. Figure 6.4 illustrates the XZ-

plane in which the vertices and polygons of the grid are generated. By setting the heights or the y-coordinates of each vertex, 3-D looking structures can be created.

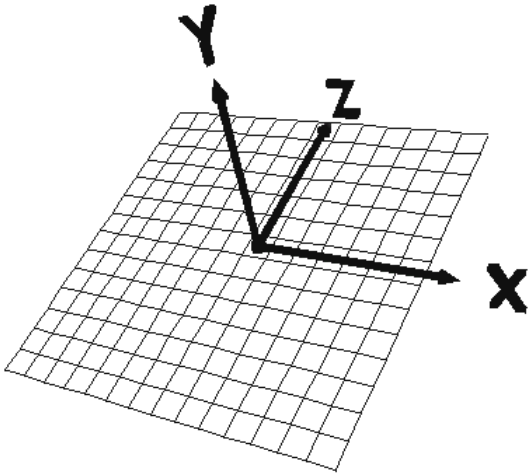

Figure 6.4 – A 3-D grid in the XZ-plane.

Pseudo code for the generation of a 3-D grid follows. Note that the variables u and v are normally used for setting up texture mapping coordinates, just like the variables x, y and z are normally used for a 3-D coordinate. Figure 6.5 illustrates how the texture mapping coordinates (u,v) are used to show a computer image on polygons. The three pictures of the tree on the left hand side are all the same. These are the original computer images of the tree. Three different sized grids are shown. The 3-D coordinates of the vertices of the polygons of the grids could be anything, for example the lower right corner vertex of the first grid could be at (100,-150,0). At each vertex in the grid, a texture map coordinate (u,v) must be specified so that the render engine can know how to stretch the computer image over the polygons of the grid. The (u,v) coordinate is a 2-D coordinate that specifies a certain position on the computer image. In Figure 6.5 the original image size is 128x128 pixels. Therefore the (u,v) coordinate (0,0) is specified at the upper left corner vertex of each grid and (128,128) is specified at the lower right corner vertex of each grid. The (u,v) coordinates of the middle vertex of each grid are (64,64). Note that the position coordinates (x,y,z) of the middle vertices are not the same, for example the position coordinates of the middle vertex of the first grid in Figure 6.5 might be (45,75,0) and that of the last grid (80,-130,0).

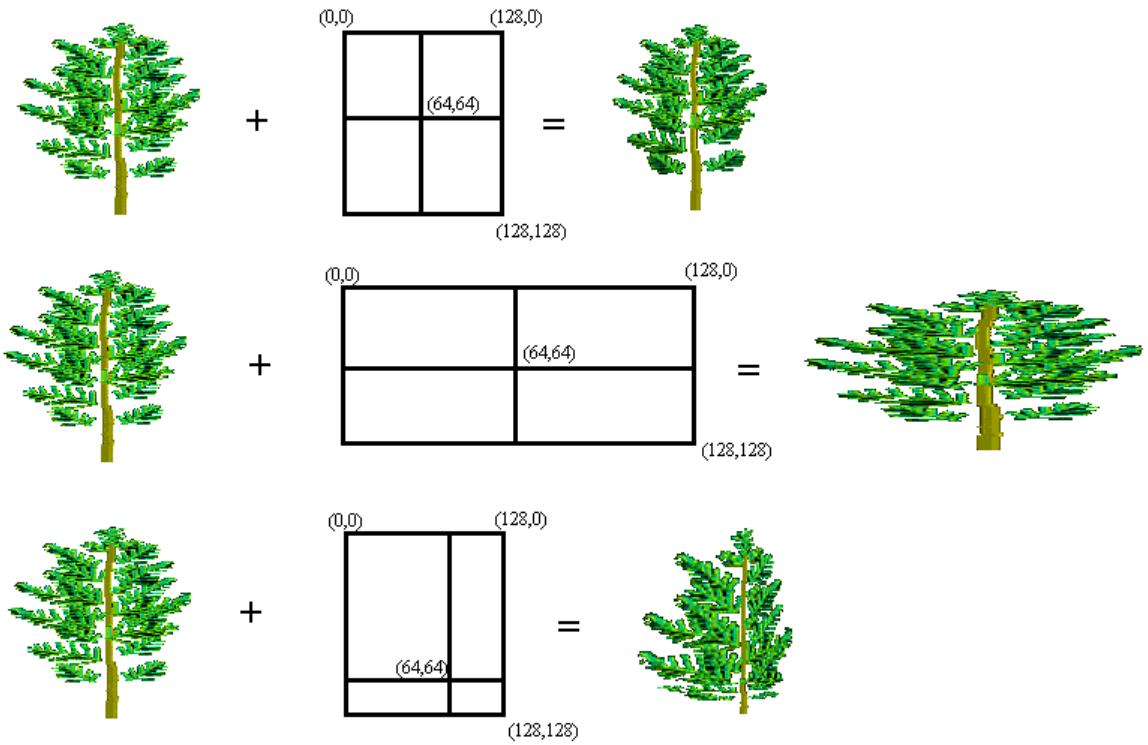

Figure 6.5 – Illustrates how the texture mapping coordinates (u,v) are used to display a computer image on polygons.

Pseudo code for the generation of a 3-D grid:

```
1. Set the grid's properties, like colour and shininess.
2. x = -1.0                    {generate vertices between –1 and 1}
3. y = 0.0                              {height is 0}
4. for CntX = 0 to GridSize-1           {horizontal}
 1. z = -1.0
 2. for CntZ = 0 to GridSize-1          {vertical}
  1. uv.u = CntX/(GridSize-1.0)         {for texture mapping}
  2. uv.v = CntZ/(GridSize-1.0)
  3. Create vertex using x, y, z, uv
  4. Add new vertex to grid
  5. z = z + 2/GridSize
 3. x = x + 2/GridSize
5. nr = 1                              {now create polygons}
6. for CntX = 1 to GridSize-1
 1. for CntZ = 1 to GridSize-1
  1. Create polygon using vertices with indexes: nr +1, nr, nr+GridSize,
                                                 nr+GridSize+1
  2. nr ++                             {row counter}
 2. nr++                               {column counter}
```

Pseudo code lines 2 to 4 show the generation of the vertices and lines 5 to 6 show the generation of the polygons. The generation of the vertices is a simple generation of a 2-D grid between –1.0 and 1.0 on the X- and Z-axes. When a vertex is added to the grid at pseudo code line 4.2.4, the vertex is assigned an index by the render engine. These vertex indexes are needed to generate polygons and to recall the coordinates of a specific vertex when needed. Line 6 shows the generation of the polygons of the grid, using specific vertex indexes.

An example of generating a 3x3 grid follows: Because it is a 3x3 grid, the grid size is equal to 3. Nine vertices will be generated between x = -1.0 and x = 1.0, and z = -1.0 and z = 1.0. When each vertex is generated and stored into memory, it is assigned a unique index number which can be used to recall the values of a specific vertex. Figure 6.6 illustrates how these nine vertices are generated.

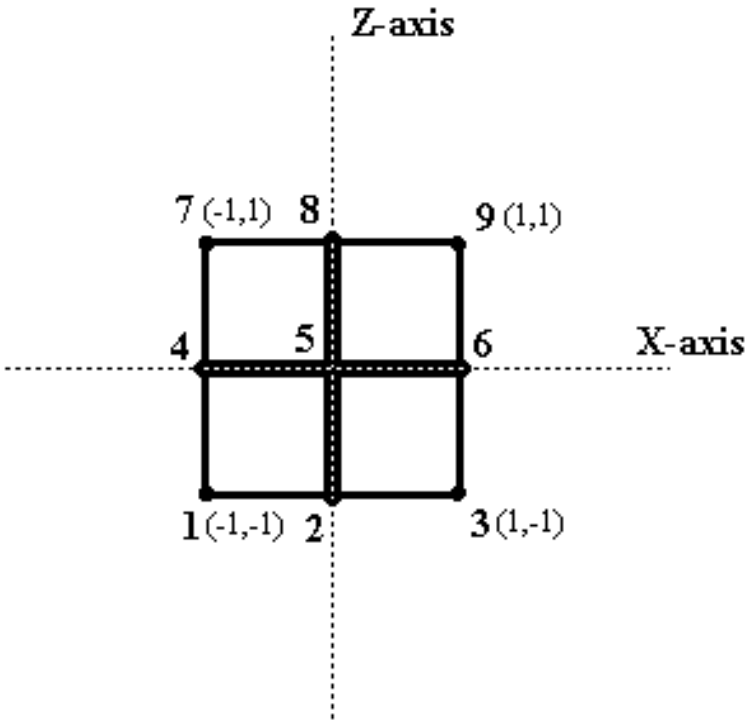

Figure 6.6 – Illustrates nine vertices generated between x = -1.0 and x = 1.0, and z = -1.0 and z = 1.0, to form four polygons.

The four polygons are formed using the following vertex indices:

Polygon 1 (2,1,4,5)

Polygon 2 (3,2,5,6)

Polygon 3 (5,4,7,8)

Polygon 4 (6,5,8,9)

6.5.2.4 3-D Condition

A *3-D condition* consists of a rectangular 3-D grid of polygons in the XZ-plane with a bitmap mapped onto it. An existing *3-D condition* can either be loaded or a new one can be generated. A bitmap can be loaded and mapped onto the grid. This will make the grid look like a flat 2-D condition in 3-D space. Plate 6.8 illustrates a flat 3-D grid with a computer image of Gastritis applied to it. The current *3-D condition*'s structure or shape can be manipulated via the *mathematical function grid manipulator*.

6.5.2.5 Mathematical Function Grid Manipulator

The output of the *3-D grid* generator is the *3-D condition*. When a new condition is generated, the condition is a flat grid of polygons. This module is used to change the shape of the *3-D condition*.

This module uses certain predefined mathematical functions, for example the function for a circle $x^2 + y^2 = r^2$, to manipulate the shape of the *3-D condition*. The user can select a certain function and set its parameters via the *computer graphic user interface*. The function $f(x)$ is applied to the y-coordinate of each vertex of each polygon in the grid, thus altering the height of each vertex in the XZ-plane. Plate 6.9 shows a 3-D grid that has been altered using a mathematical function.

By applying $f(x)$ to the grid, very smooth objects can be created. Most of the abnormal gastrointestinal conditions are, however, not smooth. Random noise can be added to each vertex's height to distort the smooth shape of the condition. Plate 6.10 shows a 3-D condition with random noise added to its vertices' heights. The following sections will discuss in detail how mathematical functions are used to alter the shape of a 3-D condition.

In order to simulate an abnormal condition that appears three dimensional, 2-D mathematical functions can be applied to the generated flat grid. Using the origin (0,0,0) as the centre of the grid, the distance to each vertex in the grid can be calculated. This distance can be used as input for the function *f*(x). The height or y-coordinate of each vertex is then set to the value of *f(x)*.

After studying the shapes of different real abnormal conditions, it was recognised that most of the abnormal conditions can be approximated using a hemispherical-like structure. Several hemispherical-like functions exist, like the function for a circle or parabola. Using some scaling factors and random distortion, these functions could be used very effectively to simulate most abnormal conditions. At the moment two functions are implemented:

1) Circle function

$$f(x) = \sqrt{radius^2 - x^2} \qquad (6\text{-}1)$$

The perfect hemisphere is a half-circle which can be obtained using equation (6-1). Therefore this was the obvious first choice of functions to use. Because a scaling factor will be applied to the function by the user, the computer graphic user interface refers to this function as an elliptic function rather than a circle. Most conditions can be simulated using this function, by setting the scaling factor and adding random distortion for a more organic look.

2) Butterworth function

$$f(x) = Scale * \frac{1}{(1 + x^2 a^2)} \qquad (6\text{-}2)$$

with $$a = \frac{Gradient}{Radius} \qquad (6\text{-}3)$$

Although most conditions can be simulated using the circle function, it was thought to be wise to add at least one more function for the user to choose from. This gives the user some variety in generating 3-D conditions. The parabola function was considered, but since the results of the elliptic function (after applying random distortion) closely resembles that of the parabola, the Butterworth function was rather chosen. The

Butterworth function is used in image processing with image enhancement in the frequency domain [GON93].

While doing research on image processing, a 3-D perspective plot of the Butterworth low pass filter proved to be very useful as a function in the 3-D condition generator. A low pass filter can be used to blur an image by reducing the high-frequency content of the image. Interesting 3-D shapes can be obtained using the Butterworth function with the scaling factor, random distortion and the "Shift" scroll bar. By setting the "Gradient" scroll bar, the function can be set to look "sharper" or "fatter". In future more functions will be added to the system. Refer to the movie clip "**\Movie Clips\3-D Condition Generator\2-Editing a 3-D Condition.avi**" on the CD-ROM for an illustration of what the Butterworth function looks like and how it can be used with the 3-D condition generator.

An example of how a 3-D grid's shape is altered follows:

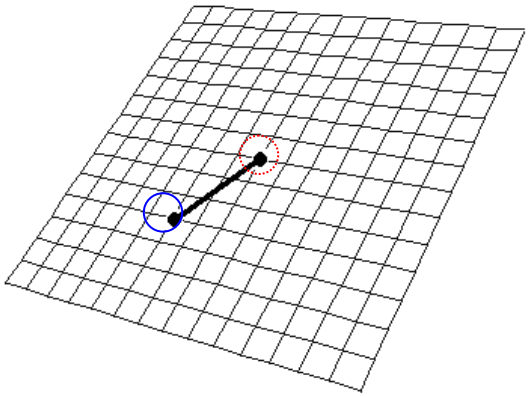

Figure 6.7 – A 3-D grid with its centre vertex indicated by the red circle and another vertex indicated by a blue circle.

To calculate the height or the y-coordinate of the vertex indicated by the blue circle on the grid in Figure 6.7, using the function:

$$f(x) = \sqrt{Radius^2 - x^2} \qquad (6\text{-}4)$$

which is shown in Figure 6.8, the distance from the marked vertex to the centre vertex can be calculated, using the distance formula:

$$Dist = \sqrt{(Centre.x - Selected.x)^2 + (Centre.z - Selected.z)^2} \quad (6\text{-}5)$$

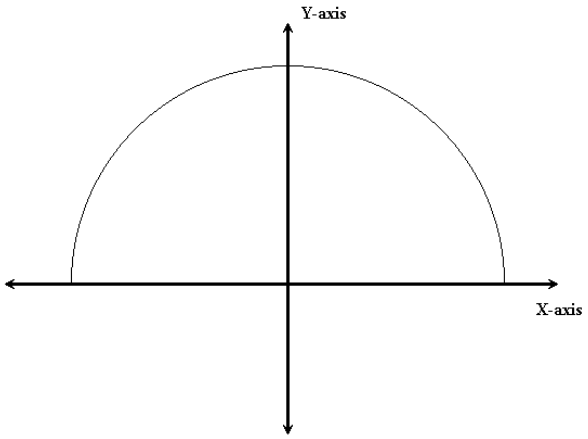

Figure 6.8 – The graph of the positive half circle *f(x)*.

This distance can now be used as input for *f(x)*:

$$f(Dist) = \sqrt{Radius^2 - Dist^2} \quad (6\text{-}6)$$

It must be checked that the distance is not greater than the radius, otherwise *f(Dist)* will be invalid since the square root of positive numbers only can be calculated. If such a case occurs, the height is set to 0. The marked vertex's y-coordinate must be set to the calculated value of *f(Dist)*. Doing this with all the vertices in the grid will create a 3-D structure as shown in Plate 6.9.

After applying a mathematical function to the 3-D grid, the result is a perfectly smooth and symmetric 3-D structure. Most of the abnormal conditions are not perfectly smooth and do not have a perfect symmetric shape, for example ulcers. The solution to this problem was to let the user make use of random noise to distort the shape of the 3-D structure. Using a scroll bar, the user can set the maximum amount of distortion and the radius of distortion. Each time the height of a vertex is set using the "Dent" scroll bar, a random number is generated for each vertex in the grid, that will be used to alter the height of each vertex.

This random number can be a positive or negative number, which means that the distortion can cause the height of the vertex to increase or decrease. Note that only vertices within the distort radius will be affected. Using the distort function makes the generated 3-D conditions look very life-like and more organic. Plate 6.10 illustrates a generated 3-D condition with random noise.

Pseudo code for applying mathematical functions to a 3-D grid follows:

```
1. if function Radius = 0
  1. a = Gradient / 0.001
   else
  2. a = Gradient / function Radius
2. middle_index = GridSize * (GridSize / 2 ) + 0.5
3. Get middle vertex, using middle_index
4. for cnt = 0 to NumVertices-1
  1. Get vertex, using cnt+1 as index
  2. Distance = sqrt ( (middle.x-vertex.x)² + (middle.z-vertex.z)² )
  3. if Distance <= distort Radius
    1. random = (rand() / MAXIMUM_RAND) * UserDestort
     else
    2. random = 0
  4. if Distance <= function Radius
    1. if f(x) is the "Butterworth" function
      1. fx = Dent / (1.0 + (Distance*a)² )
       else
      2. fx = Dent * sqrt(Radius² - Distance²) / Radius
    2. vertex.y = fx + random - (random/2.0)
     else
    3. vertex.y = 0
```

Pseudo code lines 1.1 and 1.2 prevent division by zero when the variable “a” is calculated for the Butterworth function. Referring to Figure 6.7, the coordinates of the centre vertex have to be found since all distance calculations will be relative to the centre vertex. Lines 2 and 3 get the coordinates of the centre vertex in the variable “middle”. Line 4 steps through all the vertices in the grid. The coordinates for each of these vertices are stored in the variable “vertex” (line 4.1). Then the distance to the “middle” vertex is calculated (line 4.2). Line 4.3 checks if the current vertex is within the distort radius and if it is, a random number is generated, otherwise the random number is zero. The last part of the algorithm starting at line 4.4, checks if the current vertex is within the function radius and if it is, uses the currently selected function *f(x)* and the random number to change the height or y-coordinate of the current vertex. If the current vertex is not within the function radius, then the height is set to zero. Adding these 3-D conditions to the VR model will be discussed in the next chapter.

6.5.2.6 Render Engine

The *render engine* takes as input the *3-D condition* and calculates the 2-D image from the 3-D camera's viewpoint, making use of the positions of the camera and light source(s) in the scene. The output is then relayed to the *computer graphic user interface* so that it can be displayed on the *computer screen*.

6.6 The 3-D Condition Database

The database consists of a text file with information about each condition. Each record in the 3-D condition database has the following structure:

Condition Name
Description of Condition
Filename of 3-D Condition

Condition Name is a string that represents the name of the condition. *Description of Condition* is a string that represents a description of the condition. *Filename of 3-D Condition* is also a string that contains the name of a file with the computer-generated 3-D condition. For example:

Hyperplastic Polyp
Smooth overlying mucosa resembling that of normal stomach.
Polyp.con

Since no complex queries will ever be done on the database, the database was implemented using a text file. In future it might be considered to implement the database as a "real" database, like a paradox database.

6.7 Conclusion

The purpose of this chapter was to explain what the 3-D condition database is and how the 3-D conditions are generated. The main problem addressed in this chapter is a method of simulating abnormal conditions as realistically as possible. Two methods were examined.

The first method involved using processed computer images as textures on polygons of the VR model. The second method involved the generation of a 3-D grid and applying a processed computer image onto the 3-D grid. The second method proved to be much more realistic, since most abnormal conditions are not flat structures, but 3-D. The advantage of the first method is that no extra vertices have to be added to the VR model. Only the texture of a polygon in the VR model needs to be changed. Therefore the rendering speed is not effected. The disadvantage of the 3-D conditions are that for each 3-D condition added to the VR model, extra vertices and polygons need to be added to the VR model, which effects the rendering speed negatively. The advantage of the 3-D conditions is that they can be simulated much more realistically. To give the 3-D conditions a more organic and realist look, random distortion can also be added to the vertices of the 3-D grid.

Referring to Figures 4.1, 5.1 and 6.1, the whole system has been explained except for the virtual reality simulator itself, the product that the end user will work with. By now the reader should have a good understanding of the three main data sets and other main components of the system. In the next chapter the virtual reality system will be discussed.

Chapter 7

The Virtual Reality System

7.1 Introduction

This chapter will discuss the virtual reality system, which is the software application that the end user (medical trainees and doctors) will work with. The system uses three main data sets, the VR model (refer to Chapter 4), the region database (refer to Chapter 5) and the 3-D condition database (refer to Chapter 6), to simulate certain endoscopic procedures.

Section 7.2 will explain the different user interfaces of the virtual reality system. Section 7.3 will explain how to use the virtual reality system from an end user's point of view. Section 7.4 will be used to discuss the design and implementation aspects of the virtual reality system. All the main components of the system will be discussed in detail. Pseudo code for most of the components will also be shown.

7.2 The Virtual Reality System's User Interfaces

To interact with the system, the user must make use of a user interface. This system's user interface consists of two parts. The first part is a physical user interface and the second part is a computer graphic user interface. The two user interfaces can exist and function independently of one another. The following two sections will explain these two user interfaces.

7.2.1 Physical User Interface

The physical user interface consists of a real gastroscope and a physical model of the stomach. A real gastroscope is used so that the trainees can learn how to operate a gastroscope. The gastroscope's front tip is slid into the physical model's esophagus. A trainee can then slide the gastroscope further into the model of the stomach to practice certain manoeuvres inside the stomach. Refer to movie clips "**\Movie Clips\4- Gastroscope in Physical Model.avi**" and "**\Movie Clips\5- Gastroscope in Physical Model (close up).avi**" on the CD-ROM.

The physical user interface can exist and function independently of the computer graphic user interface. This means that a trainee can practise certain manoeuvres, without the computer being switched on, using the physical model of the stomach and the real gastroscope. Note that the physical model is mounted so that the patient is lying on his/her left side, the same orientation used for a real procedure on a real patient.

The gastroscope's main function will be to serve as an input device. Just as a user will hold a computer mouse as an input device in his/her hand and move it around, the user holds the gastroscope in his/her hand. The user can then do all the manoeuvres that can be

done in a real patient's stomach, like bending the tip of the scope using the correct controls, or sliding the scope through the pylorus into the duodenum. In future a real therapeutic tool, like a biopsy tool, can also be added as another input device. The user can then slide the therapeutic tool down the instrument channel of the gastroscope when practising of therapeutic procedures is required[4].

The main function of the physical model of the stomach will be to serve as the output device. The question is what it gives as output. The answer is force feedback. Just as a computer screen serves as an output device for a user to *see* certain things, or a speaker serves as an output device for a user to *hear* certain sounds, the physical model serves as an output device for a user to *feel* force feedback. It is also an output device for the user to *see* where the tip of the gastroscope is inside the physical model.

The following section will explain what force feedback is, how it can be used and how it is used in this system.

7.2.1.1 Force Feedback

Force feedback is the term used to describe a force that a person feels when touching or pushing against an object. Applied to this system, when the user touches the side of the organ with the gastroscope, it has to feel as if he/she is touching the side of a real organ.

Some expensive devices, like the joystick of Immersion Corporation, use simple force feedback. Some of the more expensive VR data gloves, like the Cyber Glove, actually have small touch plates built into the fingertips of the glove. These can be activated by software to press against the user's fingertips so that the user feels as if he/she has touched something. For example, if the user were in a virtual world and had to pick up a certain object, he/she would reach out to the object. The 3-D tracking device will be used to know where the user's hand is. As soon as the user touches the object, the sensors will be activated to press against his/her fingertips. This will feel as if he/she is touching the object, even though there is no real object, only the one in virtual space. Because force feedback is very difficult to implement, it is also very costly.

[4] This feature has already been implemented, but will not be discussed as part of this thesis.

Immersion Corporation has developed force feedback systems for cutting tissue with a scalpel and injecting needles into certain tissues. A minimally invasive surgery simulator exists that uses some of these force feedback systems. These systems are all very expensive. To simulate force feedback in the system described in this document, a very inexpensive method is used, namely natural force feedback. The term natural force feedback describes force feedback that does not have to be simulated by software, but is induced physically. For most VR applications this is not possible since the virtual worlds' objects can be placed and moved anywhere. With this system, the main object in the virtual world is the VR model of the stomach. The stomach is a stationary organ, therefore the physical model can be used for natural force feedback. This version of the system works with a physical model made of fibreglass, but a real stomach's elasticity is definitely not the same as that of fibreglass. Therefore to make the force feedback more realistic, a rubbery material with almost the same elasticity as a real stomach will have to be used for the physical model[5].

The advantages of using natural force feedback are the following. First of all it is much cheaper to implement than to implement with actuator-controlled force feedback. Secondly, the physical model will restrain the user to do certain things which will be impossible using actuator-controlled force feedback. For example, with this system, the gastroscope should not be able to penetrate the stomach. Looking at it physically, the gastroscope is restrained to go outside the physical model of the stomach, unless the fibreglass is broken and the gastroscope can penetrate the physical model. Because the physical position and orientation is relayed to the system and the VR model is registered with the physical model and warping is implemented, the tip of the virtual gastroscope will never penetrate the VR model. The restrained volume naturally takes the shape of the physical model. To implement this using actuator-controlled force feedback, is almost impossible. The disadvantage of natural force feedback is that it can only be used in very specific applications where the physical models must be used.

[5] At the time of this writing the current system is using a silicon physical model. This was, however, regarded as beyond the scope of this study and is therefore not included in this thesis.

7.2.2 Computer Graphic User Interface

The system is a Microsoft Windows 95 platform application, which means that most of the usual Windows 95 user interactions are possible, for example, moving, resizing, and closing windows.

The computer graphic user interface can exist and function independently of the physical user interface. The computer graphic user interface's input and output devices are all computer/electronic devices and therefore do not depend on the physical user interface. For example, the inside of the VR model can still be viewed by positioning the 3-D tracker correctly without it being fitted onto the tip of a gastroscope.

The keyboard and mouse serve as input devices for standard input. The 3-D tracker serves as an input device with which the system calculates the position (x, y, z) and orientation (yaw, pitch, roll) of the receiver, so that the appropriate viewpoint can be calculated. It is not part of the physical user interface because it is a computer/electronic input device just like the mouse and keyboard.

The computer screen serves as an output device on which the user sees different windows and menus. The main window is used to display the calculated viewpoint, mentioned in the previous paragraph, that simulates a real stomach on the inside.

The following section will explain how to use the virtual reality system. All the menu items and functionality will be explained. Chapter 8 gives two examples of how instructors can use some of these functions to set up tests for trainees.

7.3 Using the Virtual Reality System

It is suggested that when a trainee has never worked with a gastroscope before, that the multimedia system is worked through first. This will ensure that the trainee knows how to hold the instrument, how to manoeuvre it and how to move the tip of the gastroscope using the controls. When the system starts, the computer graphic user interface appears. This is the main window of the application. Plate 7.1 shows the main window. The main menu is

at the top of this window and at the bottom is a message bar that shows certain messages to help the user. The part of the window that will show what the simulated gastroscope's view looks like, is between the main menu and the message bar.

After the system has started, the user can start exploring the inside of the stomach by moving the tip of the gastroscope down the esophagus and into the stomach. Thus, without using any of the menu options, a user can already start to experience the feeling of doing a gastroscopy. Refer to the movie clips "**\Movie Clips\4- Gastroscope in Physical Model.avi**" and "**\Movie Clips\VR System\4- Orientation and Organ Windows.avi**" on the accompanied CD-ROM to get an idea of how the system works. As the real gastroscope is slid in and out of the physical model (first movie clip), a viewpoint is rendered that looks like a real stomach inside and corresponds to the movement of that of the real gastroscope (second movie clip). In the second movie clip three windows are visible. The main window and camera orientation window correspond to the movement of the real gastroscope. The organ window can be used to identify certain anatomical regions of the stomach. This will be discussed later in the section.

When the system starts, it is in *learn mode*. Using the "Options" menu item, the instructor can set the system to *test mode* so that a trainee can be evaluated. To be evaluated, a trainee must first be logged onto the system with a name and number. This information will be used in the evaluation report.

One of the aims of this system is to teach a trainee basic identification skills. Abnormal conditions can be added inside the stomach by either loading an identification training data file or by using the edit menu option. Another aim of this system is to teach a trainee basic navigation skills. Navigation training data files containing navigational tasks can be loaded or defined using the edit menu option. The training of identification and navigation skills is done separately. The trainee must choose from the "Train" menu item which training he/she wants to do.

The system's menu will now be discussed:

File

User

Allows the user to log onto the system with a logon name and number, for example trainee name and number. This will also allow the user to log off again or get an evaluation of tests completed. If the user is not logged on, no evaluation can be done. In other words the Test Mode cannot be used. At this stage the system does not provide for passwords for each user. This feature should be added to the system if the tests were to be done without any supervision.

Load Identification Training Data

Loads an existing scenario with 3-D conditions. These files contain the names and positions inside the VR model of each 3-D condition that was previously placed in the VR model when the specific scenario was saved. This information is then used to reconstruct the scenario.

Load Navigation Training Data

Loads an existing task list for navigation training. This list contains tasks that the trainee must complete, like doing biopsies.

Save Identification Training Data

Saves the current scenario of 3-D conditions. Each added 3-D condition's name and position inside the VR model are saved so that the scenario can be reconstructed when the file is loaded.

Save Navigation Training Data

Saves the current task list for navigation training.

Exit

Exits from the application.

Train

Identification Skills

Basic identification skills are to identify certain basic conditions, for example an ulcer or polyp. When checked, an instructor can add 3-D conditions to the VR model. The trainee must navigate through the VR model to identify these conditions. In *Learn Mode* a trainee can click on a condition to see a description of the condition and in *Test Mode* the trainee will be asked a multiple choice question to identify the condition.

Navigation Skills

Navigation skills are skills to do with hand-eye co-ordination, for example to take a biopsy or to cauterise a certain region. When checked, an instructor can set up specific tasks which a trainee must then perform. In *Test Mode* the system will time how long the trainee takes to complete each task and this will be included in the evaluation report.

1. Stomach

In future other organs could be added to this list, for example lungs, uterus, etc.
This is currently the default value for the system.

Edit

Edit Mode

When checked, the system is in edit mode. In edit mode, an instructor can set up certain scenarios and task lists on which the trainees can be tested. Nothing stops a trainee to also use the edit mode for setting up his/her own tests, so that he/she can practise difficult tasks on his/her own.

When training is set for identification skills, the VR model is shown using flat shading (rather than smooth shading) and the 3-D condition browser opens on the screen from which a condition can be selected. Plate 8.2 illustrates what the 3-D condition browser looks like. Flat shading shows each polygon in the VR model much better as separate polygons. Compare Plate 8.3 with Plate 8.4. Plate 8.3 shows the inside of the VR model using flat shading. Each polygon in the model

can clearly be seen so that it is easier to know where abnormal conditions can be placed. Plate 8.4 shows the inside of the VR model using smooth shading. The separate polygons cannot be seen very clearly using smooth shading. When clicking on a polygon inside the VR model, the selected condition will be added to the VR model.

When the training mode is set for navigational skills, a dialog box appears, showing all the tasks in the current task list. New tasks can be defined or existing ones can be deleted. New tasks can easily be constructed using two drop down lists. The first drop down list defines the action and the second one the region on which the action should be performed. For example, “Examine Pylorus”, or “Do Biopsy on Marked Region”.

Undo

When checked, the edit mode is in undo mode. This means that when the user clicks on a 3-D condition in the VR model, the condition will be removed from the VR model. This option is only available when in Edit Mode.

Undo All

Removes all the 3-D conditions in the VR model, to get a “clean” VR model without any conditions. This option is only available in Edit Mode.

View

Camera Orientation

Shows a window with a wireframe model of the VR model and the position and orientation of the tip of the gastroscope inside the VR model. The movement of the pointer inside this window is correlated to the position and orientation of that of the real gastroscope’s tip. Plate 7.2 illustrates the orientation window.

In this view double sided polygons are used so that the user can see all the polygons, in other words, no hidden face removal (refer to section 4.2.1.1) is used. Showing the tip of the scope is very important so that the user can realise exactly where the scope is inside the physical model. The learning of orientation skills

could be sped up because the trainee is able to see where the scope is in the physical model. Once a trainee has progressed far enough, this window should not be necessary any more.

Organ

Shows the VR model from the outside. This is mainly to teach a trainee where the different anatomical regions of the organ are. Clicking on any part of the VR model in this window will show the name of the region and show the whole region on the organ by changing the colour of the polygons that are part of that region to red. The user can also rotate the organ in any way. See Plate 7.3 for an example of organ window.

Video Recorder

Activates the video recorder functions. This works like a normal video recorder with record, playback, stop, pause and fast forward buttons. With the video recorder a trainee can record a procedure or test, and play it back for the instructor. The instructor can then analyse what the trainee has done and if necessary instruct the trainee to do the procedure in another way. See Plate 7.4 for an example of video recorder buttons. The main window is used to display the playback of the recorded procedures. Each time a frame is rendered the position and orientation of the tip of the gastroscope is saved in a file, and not the computer-generated image. Using this method creates very small files, even when long recordings are done.

Preparatory Procedures

Shows the trainee some basic information on preparatory procedures, for example sterilising the gastroscope. Text, images and video clips are used for this. The multimedia system can also be used for this purpose[6].

Images of Conditions

Shows 2-D images of real conditions. These are the same images that are used in the multimedia program.

[6] The latest version of the system does not have this functionality anymore, since it was decided not to duplicate work that can be done with the multimedia system.

Video Clips of Real Procedures

Shows video clips of real procedures. These video clips are also the same clips found with the multimedia program.

Options

Training Options

Allows the user to choose between Learn Mode and Test Mode. In learn mode the trainee will be given information and in Test Mode the trainee will be required to give information to the system which will be analysed.

Scope Options

Allows the user to set the current gastroscope's properties. The properties that can be set are the field of view and if it is a videoscope or fibrescope. Another property that could be added to be set, is whether it is a side viewing scope[7]. Mostly such an attribute will be used with duodenoscopes for ERCP procedures. The user can set up a customised scope by entering the values for the properties of a scope, for example the field of view, or he/she can choose a predefined scope from the menu. It is important to have this option, especially because of the field of view that can differ from 100 degrees to 140 degrees. To make the simulation as realistic as possible, the user would set up the scope so that it has the same properties as that of the one that he/she will use in a real life situation.

General Options

If for some reason some of the application windows are missing or out of bounds of the screen, the user can reset the application desktop to the default values. This will resize and reposition all the windows used in the application to their defaults. To do this, the user just has to click on the "**Reset Application Desktop to Default**" menu item.

[7] Because the latest version of the system already makes provision for ERCP procedures, this property has already been added.

When the "**Save Application Desktop on Exit**" option is checked, the size and position of all the windows of the application will be saved when the user exits the program so that these values can be used when the program is run again.

The user can also remove the status line at the bottom of the main window, but this is inadvisable since most of the user interface messages will be displayed on the status line. The "**Show Status Window**" is also a menu item that can be checked or unchecked.

Help

Help

Shows a standard Windows help file for this application.

About

Shows information about this software application.

Chapter 8 could be worked through to understand how some of the basic functions of the system works. Chapter 8 was written with the view of being independent of its preceding Chapters.

7.4 Overview of the System's Main Components

Figure 7.1 shows a diagrammatic representation of the main components of the virtual reality system and the data sets used by the virtual reality system. The following sections will discuss these components and data sets.

7.4.1 Initialisation

This module initialises certain parts of the system. This module basically prepares the system so that it is in a state of readiness to start simulating medical procedures. All the initialisations will be discussed, but normalising the VR model, smoothing the VR model and calculation of the warp detection distance will be discussed in detail.

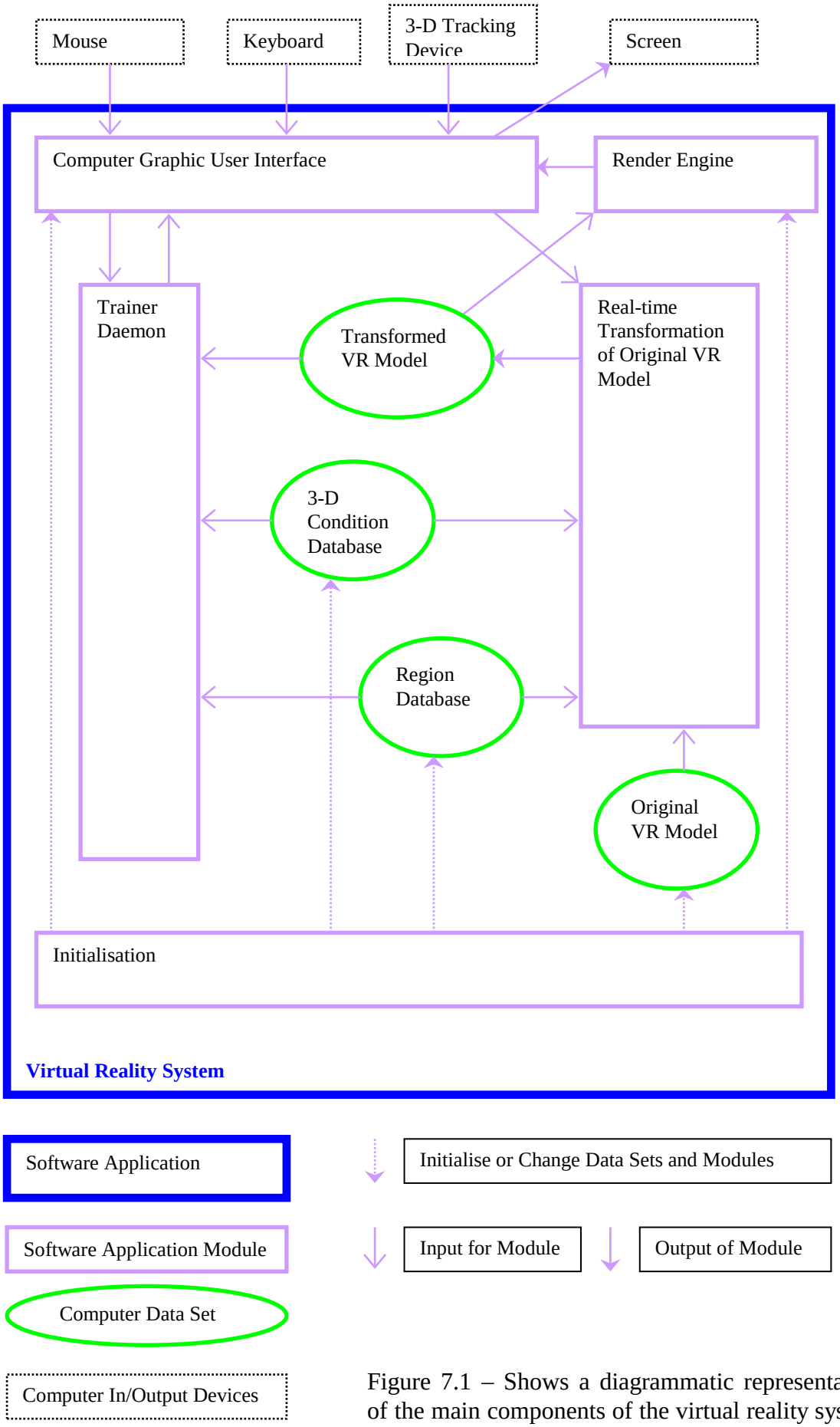

Figure 7.1 – Shows a diagrammatic representation of the main components of the virtual reality system and the data sets used by the virtual reality system.

One of the first things that has to be checked is whether the computer's desktop settings are correct for the system. Currently the system must run on a computer with its desktop set to 800x600 in 16-bit high colour mode. Using default values and values gathered from the Windows 95 registry, the system's windows are placed and sized on the desktop. Attributes for predefined endoscopes are initialised, for example the field of view.

All the 3-D initialisations are executed in this module. The *render engine*, in this case Renderware, has to be initialised before any of its functions can be used. A scene has to be created. A virtual camera has to be created and initialised to show the *VR model* from the viewpoint of the tip of the gastroscope. The previously selected gastroscope is retrieved from the Windows 95 registry and the virtual camera's field of view is set accordingly. A light source has to be created to show light reflecting from the *VR model* in the scene, otherwise the camera image will just be dark. Different kinds of light sources may be used, for example a point light, directional light or conical light (spotlight). As the type of light used on the tip of a gastroscope resembles a point light, a point light was chosen to make the simulation as realistic as possible.

The *VR model* is loaded into the scene. The *VR model* must be normalised, which involves translating the model to the origin (0,0,0) of the scene and scaling the model to fit in the unit cube. Refer to section 7.4.1.1 for a detailed discussion on the normalising of the VR model. For the render engine to render the VR model using smooth shading, the average normal vector for each vertex in the VR model has to be calculated. Refer to section 7.4.1.2 for a detailed discussion on the smoothing of the VR model.

The *VR model* is a dynamic model in the simulator, which means that its form/shape can change. Therefore the initialisation module has to make a copy of the original VR model so that the VR model can be reset to its original form/shape when needed. This copy is called the *original VR model*.

The system also needs to know how close the virtual camera can be to the surface of the *VR model* before the VR model has to be warped so that the camera cannot penetrate the VR model. This distance will be referred to as the *warp detection distance* and must also

be calculated during initialisation of the system. For a detailed discussion on how this distance is calculated, refer to section 7.4.1.3.

The *3-D condition database* is loaded into memory. Each of the 3-D conditions has a file associated with it that represents the 3-D data. This 3-D data is loaded for each condition and another scene is created with these 3-D objects for showing the conditions when in edit mode. The latter scene is used in the condition browser where the instructor can choose a condition to add to the *VR model*. Note that the system loads all the 3-D conditions into memory. This might seem a waste of memory, but for the condition browser to display all the available conditions, all of the 3-D conditions have to be in memory. The database of 3-D conditions should also not grow to have thousands of different conditions, since these are computer-generated 3-D conditions and it would be best to have just a few basic conditions for identification skills. The multimedia system can rather be loaded with thousands of images of conditions for training diagnostic skills.

The *region database* is also loaded into memory. This information will be used by other modules in the system to, for example show the user where a certain region in the *VR model* is.

The virtual therapeutic tools, like the biopsy tool, must be created. Currently the system allows for biopsy taking and cauterisation. For biopsy taking, biopsy forceps are needed and for cauterisation a heat probe is needed. These tools are also 3-D objects that are created from other basic objects, like cylinders and spheres. Plate 7.5 illustrates how a biopsy tool was created. A cylinder with a cone at the one end forms the sharp tip. Two half spheres were added to the cylinder in such a way that the half spheres can be opened and closed to simulate the grasping of something.

Lastly the 3-D tracking device has to be initialised. A Polhemus tracking device is used with this system. Using the base I/O port of the device, data can be read and written to the port. Certain initialisation characters need to be written to the port. The device is initialised to use centimetres rather than inches for this system. If the tracker cannot be initialised, the system will quit with an appropriate error message.

The Windows 95 registry was mentioned a few times in this section. The Windows 95 registry is basically a local database for each personal computer that has information about the computer's hardware, software and users. Instead of using initialisation files (.ini), as previous versions of MS Windows did, the registry can now be used to save and retrieve certain information. This application uses the registry to save and retrieve values for the position and size of most of the windows used in the system. This means that a user can place and size the windows as he/she pleases and the next time the application is used, these settings will be applied. Default values, like the field of view of a gastroscope, are also saved in the registry and used during initialisation.

The next three sections are very detailed discussions of how the VR model is normalised and smoothed during initialisation. The warp detection distance is also discussed in detail.

7.4.1.1 Normalisation of the VR Model

During the initialisation of the system, the VR model must be normalised. Because a number of different components with different scales are involved in this system, it is much easier to scale all measurements to unit space and do all calculations in unit space. For example, the measurements of the physical model and VR model will never be 100% the same. Therefore scaling the VR model to unit space and scaling the 3-D position (x, y, z) input from the 3-D tracker to unit space make other calculations easier. Normalisation involves translating the model to the origin (0,0,0) of the scene and scaling the model to fit into the unit cube, but maintaining the ratio between the x-, y- and z-coordinates.

Translating the model to the origin involves calculating the centre point of the model which is done first by calculating the minimum and maximum x-, y- and z-coordinates in the model, then adding the minimum and maximum values together and dividing it by 2. For example, if the minimum x-coordinate is –4 and the maximum x-coordinate is 6, then the centre point's x-coordinate will be 2/2=1. A 3-D vertex is obtained with coordinates in the middle of the VR model. The VR model has to be translated so that this centre point is at the origin (0,0,0). This is done by the following three calculations for each vertex in the VR model:

$$Vertex.X = Vertex.X - Centre.X \tag{7-1}$$

$$Vertex.Y = Vertex.Y - Centre.Y \tag{7-2}$$

$$Vertex.Z = Vertex.Z - Centre.Z \tag{7-3}$$

To scale the VR model into the unit cube means to scale each vertex so that each co-ordinate of each vertex of the VR model is between –1.0 and 1.0 for x, y and z. The ratio between the x-, y- and z- coordinates of the VR model has to be maintained, therefore the vertex with the largest coordinate (x, y or z) has to be calculated and each coordinate of each vertex scaled by this factor:

$$Vertex.X = \frac{Vertex.X}{MaxCoord} \tag{7-4}$$

$$Vertex.Y = \frac{Vertex.Y}{MaxCoord} \tag{7-5}$$

$$Vertex.Z = \frac{Vertex.Z}{MaxCoord} \tag{7-6}$$

where

MaxCoord = Maximum Absolute Coordinate Value of all Coordinate Values in VR Model

In other words the value –10 would be calculated larger than 5 because the absolute of –10 is 10. *MaxCoord* can therefore be the largest x, y or z value. Figure 7.2 shows the unit cube to which the VR model is scaled. Note that the shape of the VR model does not change when scaling it to unit space.

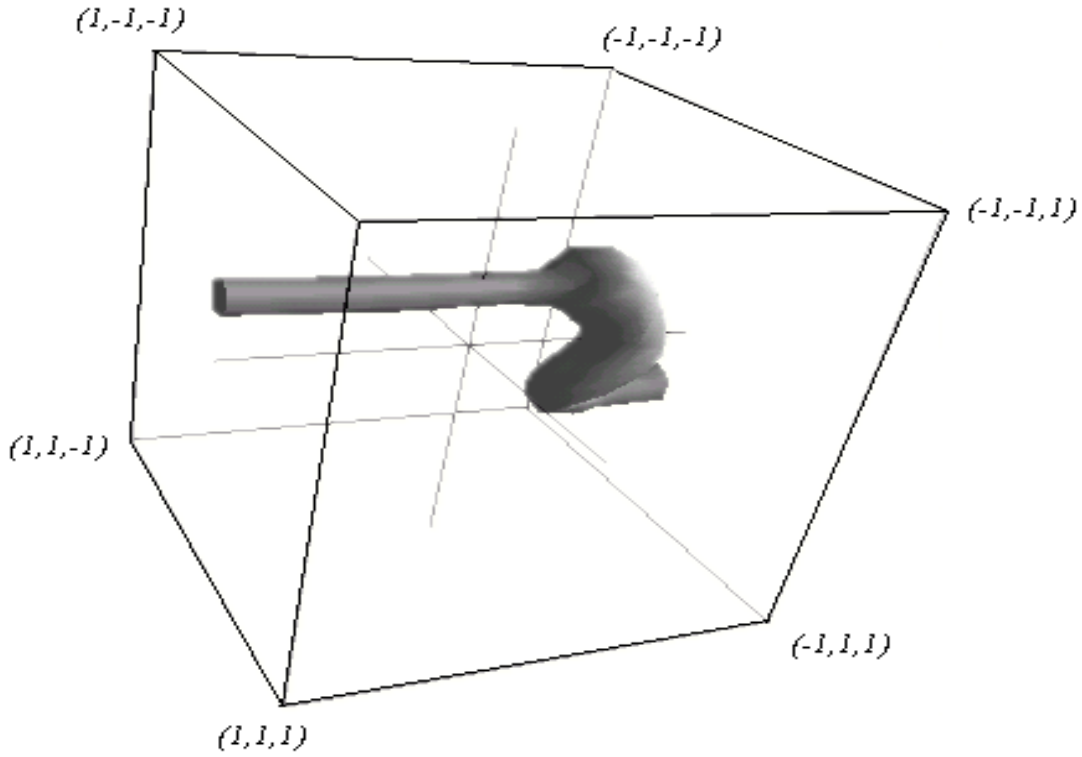

Figure 7.2 – Shows the unit cube to which the VR model is scaled.

The pseudo code for the normalisation of the VR model follows:

```
1. min.x = 3200                {ensures that a minimum value is obtained}
2. min.y = 3200                {from the first test value}
3. min.z = 3200
4. max.x = -3200               {ensures that a maximum value is obtained
5. max.y = -3200               {from the first test value}
6. max.z = -3200
7. num_vertices = Get number of vertices in model
8. for cnt = 0 to num_vertices-1    {Get min/max values for moving}
 1. Get vertex, using cnt+1 as index
 2. if vertex.x < min.x
  1. min.x = vertex.x
 3. if vertex.y < min.y
  1. min.y = vertex.y
 4. if vertex.z < min.z
  1. min.z = vertex.z
 5. if vertex.x > max.x
  1. max.x = vertex.x
 6. if vertex.y > max.y
  1. max.y = vertex.y
 7. if vertex.z > max.z
  1. max.z = vertex.z
9. for cnt = 0 to num_vertices-1          {move vertices to center}
 1. Get vertex, using cnt+1 as index
 2. centre.x = Div((min.x-max.x),2)-min.x
 3. centre.y = Div((min.y-max.y),2)-min.y
 4. centre.z = Div((min.z-max.z),2)-min.z
 5. vertex.x = vertex.x – centre.x
 6. vertex.y = vertex.y – centre.y
 7. vertex.z = vertex.z – centre.z
 8. Set vertex, using cnt+1 as index
```

```
10. min.x = 3200        {min/max values have now changed so recalculate}
11. min.y = 3200
12. min.z = 3200
13. max.x = -3200
14. max.y = -3200
15. max.z = -3200
16. for cnt=0 to num_vertices-1 {Get min/max values for scaling}
 1. Get vertex, using cnt+1 as index
 2. if vertex.x < min.x
  1. min.x = vertex.x
 3. if vertex.y < min.y
  1. min.y = vertex.y
 4. if vertex.z < min.z
  1. min.z = vertex.z
 5. if vertex.x > max.x
  1. max.x = vertex.x
 6. if vertex.y > max.y
  1. max.y = vertex.y
 7. if vertex.z > max.z
  1. max.z = vertex.z
17. max_coordinate = max(max (max.x,max.y),max.z)
18. min_coordinate = min(min (min.x,min.y),min.z)
19. if max (Abs(max_coordinate),Abs(min_coordinate)) = 1
 1. return                                      {already normalised}
20. scaling_factor = max (Abs(max_coordinate),Abs(min_coordinate))
21. for cnt=0 to num_vertices-1
 1. Get vertex, using cnt+1 as index
 2. vertex.x = Div (vertex.x,scaling_factor)
 3. vertex.y = Div (vertex.y,scaling_factor)
 4. vertex.z = Div (vertex.z,scaling_factor)
 5. Set vertex at index cnt+1 to vertex
```

Pseudo code lines 1 to 8 and 10 to 16 form a simple way of finding the minimum and maximum dimensions of the model. First the minimum coordinates are set to large values and the maximum coordinates are set to small values. This ensures that the first test value will already be seen as a maximum or minimum value. It is necessary to repeat this search for maximum and minimum coordinates in lines 10 to 16 since the vertices have been moved or translated in line 9. Pseudo code lines 9.5 to 9.7 correlate with equations (7-1), (7-2) and (7-3). This is where the coordinates for vertices are recalculated to be moved. Lines 17 and 18 check to see which coordinate of any vertex is the furthest away from the unit coordinates by comparing the maximum and minimum x, y and z coordinates. In line 20 the absolute maximum coordinate is found as a scaling factor. Lines 21.2 to 21.4 correlate with equations (7-4), (7-5) and (7-6).

7.4.1.2 Smoothing of the VR Model

The VR model consists of a number of flat polygons. Plate 7.6 illustrates how these polygons look if flat shading is used during rendering. It is possible to display the VR model as a smooth model using smooth shading techniques, like Gouraud or Phong shading [VIN95]. Plate 7.7 illustrates what the VR model looks like when smooth shading is used. Render engines can render a model using either flat or smooth shading, given the fact that the model was correctly generated. If the VR model is generated correctly for the render engine, it ensures that the normal vectors of the VR model can be generated *automatically* by the render engine, otherwise the normal vectors have to be calculated "manually" or no smooth shading is possible.

Because of design specifications for the VR model of this system, the render engine of this system unfortunately cannot render the VR model using smooth shading without first setting the VR model's normal vectors "manually". The design specifications for the VR model do not allow vertices to be shared between different polygons. Such vertices also have to be declared for each polygon even if they share the same 3-D space. Therefore extra calculations have to be done during initialisation to achieve smooth shading. After these calculations have been done, the VR model's normal vectors are set correctly so that the render engine can achieve smooth shading again.

This section will not explain how Gouraud or Phong shading works, but will explain how the VR model has to be prepared each time the system starts so that the render engine can make use of smooth shading, like Gouraud or Phong shading. Some general knowledge of what is needed for smooth shading will first be discussed.

Each polygon in the VR model can be triangulated so that each triangle is restricted to a single plane [AMM92, VIN95]. For each of these triangles a normal vector can be calculated using the cross product [SAL90]. The normal vector of a triangle can be used as a direction vector. Each normal vector is of unit length and perpendicular to the plane of its triangle.

Given a triangular plane with vertices P, Q and R a normal vector **norm** can be calculated [FER96, LAN86]. Firstly two vectors **a** and **b** have to be calculated using the three vertices P, Q and R.

$$\mathbf{a} = Q - P \tag{7-7}$$

$$\mathbf{b} = R - P \tag{7-8}$$

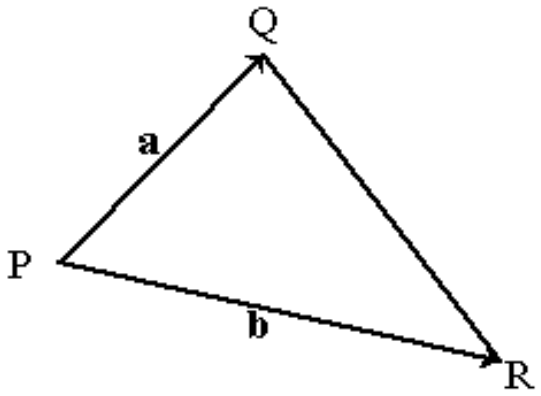

Figure 7.3 – Shows two vectors calculated from the vertices of a triangle.

These two vectors are now used to calculate a vector perpendicular to the triangle's plane, using the cross product.

$$\mathbf{norm} = \mathbf{a} \times \mathbf{b} \tag{7-9}$$

This can be calculated using:

$$norm.X = (a.Y * b.Z) - (a.Z * b.Y) \tag{7-10}$$

$$norm.Y = (a.Z * b.X) - (a.X * b.Z) \tag{7-11}$$

$$norm.Z = (a.X * b.Y) - (a.Y * b.X) \tag{7-12}$$

The vector **norm** is now perpendicular to the triangle's plane. A normal vector must be of unit length, therefore the length of the vector must be calculated and used to set the length of the vector to 1 [SAL90].

$$Length = \sqrt{norm.X^2 + norm.Y^2 + norm.Z^2} \quad (7\text{-}13)$$

$$norm.X = \frac{norm.X}{Length} \quad (7\text{-}14)$$

$$norm.Y = \frac{norm.Y}{Length} \quad (7\text{-}15)$$

$$norm.Z = \frac{norm.Z}{Length} \quad (7\text{-}16)$$

Now the vector **norm** is a normal vector for the triangle's plane. If a polygon consists of more than one triangle, an average normal vector can be calculated for the polygon using the normal vectors of each triangle. The **average normal** vector can be calculated by adding up all the triangles' normal vectors of the polygon to find a **total** vector, then dividing the **total** vector by the number of triangles in the polygon. According to the design specifications of the VR model, the polygons of the VR model in this system will always consist of two triangles.

To calculate the intensity of the light that should be reflected from a polygon, given the position of a light source, the render engine uses this normal vector of a polygon. Since a simple light model is used, it is assumed that the angle of reflecting light will be the same as the angle of incoming light. Therefore the intensity of light for each polygon can be calculated and its colour can be set according to the light intensity. When only this is done, the term flat shading is used. The results are shown in Plate 7.6. It can clearly be seen that the polygons facing the light source reflect more light than the polygons not facing the light source.

To make the VR model look smooth, the VR model can either be digitised more densely or smooth shading can be used. Gouraud shading was used in this system. A colour for each vertex is calculated using the vertex normal. The colour is then linearly interpolated across the face of the polygon. Smooth shading is used by assigning an average normal

vector to each vertex in the VR model. To calculate the average normal vector of each vertex, each polygon that shares this particular vertex has to be determined and the sum of these polygons' normal vectors are calculated. Renderware, the render engine, does all the triangulation of polygons and calculation of normal vectors for polygons and shared vertices. Note that the images in Plate 7.6 and Plate 7.7 were obtained using the same amount of 3-D vertices.

The design of the VR model specifies that each polygon in the VR model must be able to have its own texture or image (refer to section 4.2.1.3). For each polygon to have its own texture, Renderware requires that each polygon have its own vertices. The vertices in the VR model can therefore not be shared amongst polygons. This means that for each polygon, its vertices must be declared separately, which means that the average normal vectors for the shared vertices must be calculated and set by the system during initialisation. If it were not for the design specifications of the VR model, then the render engine could have rendered the VR model using smooth shading, without any other calculations. The design specifications were not changed to make the smoothing of the VR model easier because the first design specification for the VR model states that each polygon in the VR model should have its own texture so that good image resolution can be obtained.

Figure 7.4 is an example of polygons sharing vertices and Figure 7.5 is an example of polygons sharing the same 3-D points in space, but do not share the same vertices in memory.

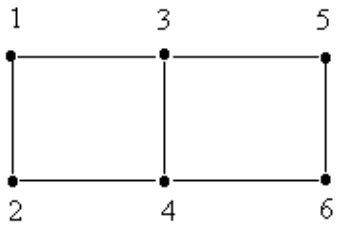

Figure 7.4 – Two polygons sharing vertices 3 and 4.

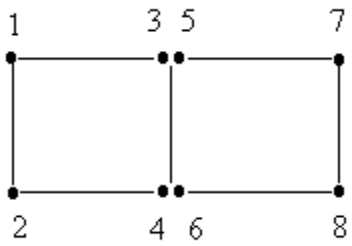

Figure 7.5 - Two polygons not sharing any vertices, although vertices 3 and 5 share the same 3-D space and vertices 4 and 6 share the same 3-D space.

The pseudo code for preparing the VR model for smooth shading is the following:

```
1. set normalized[0] .. normalized[num_vert-1] = -1
2. for cnt=0 to num_vert-1
 1. if (normalized[cnt] != 1)
  1. total.x = 0
  2. total.y = 0
  3. total.z = 0
  4. normal.x = 0
  5. normal.y = 0
  6. normal.z = 0
  7. number = 0
  8. Get cur_vertex, using cnt+1 as index
  9. for cnt2=0 to num_vert-1
   1. Get test_vertex, using cnt2+1 as index
   2. if (test_vert.x = cur_vert.x) AND
         (test_vert.y = cur_vert.y) AND
         (test_vert.z = cur_vert.z)
    1. Get cur_vertex normal vector in normal, using cnt2+1 as index
    2. total.x = total.x + normal.x
    3. total.y = total.y + normal.y
    4. total.z = total.z + normal.z
    5. number = number + 1
    6. normalized[cnt2] = 0                {found one to be normalized}
  10. Normalize the vector total
  11. for cnt2=0 to num_vert-1
   1. if normalized[cnt2] = 0
    1. Set vertex normal to vector total, using cnt2+1 as index
    2. normalized[cnt2] = 1
```

Each vertex in the VR model is checked against each other vertex in the VR model for sharing the same 3-D space. See pseudo code line 2.1.9.2. If the same 3-D space is shared, the normal vector of that vertex is added to a total vertex. After all the vertices have been checked, the total vertex is normalised. If any other vertices sharing the same 3-D space were found, then the vertex's normal vector is set to the normalised total vector. See pseudo code lines 1.2.1.11.

To summarise this section, a render engine can either render a VR model using smooth or flat shading. To use smooth shading, average normal vectors have to be calculated for each vertex by the render engine. Because of the design specifications for the VR model, the system has to calculate and set these average normal vectors for the render engine, otherwise the rendering of the VR model using smooth shading will look like rendering of the VR model using flat shading.

7.4.1.3 Calculation of the Warp Detection Distance

The warp detection distance is the minimum distance that the tip of the gastroscope may get to any polygon in the VR model before that polygon has to be warped away from the tip of the gastroscope, so that the tip can never penetrate the VR model. The calculation of the warp detection distance is therefore very important for the collision detection and warping of the VR model. Refer to section 7.4.4.1 for more detail about collision detection and section 7.4.4.2 for more detail about warping.

To be able to determine how close the tip of the gastroscope may get to any polygon in the VR model before the polygon has to be warped away from the scope, the “size” of each polygon in the VR model must be inspected.

The “size” of each polygon will be defined as the minimum radius around each vertex so that the area covered by this minimum radius will cover the whole surface of the polygon. To explain this better, see Figure 7.6. Around each vertex of the polygon a radius of the same size is drawn. It can clearly be seen that the area in green is not covered or enclosed by the size of this radius. The size of the radius must be enlarged until the whole surface of the polygon is enclosed. The size of this minimum radius that will enclose the whole surface of the polygon will be used as the size of the polygon. Figure 7.7 shows the situation where a radius is used that is too big, which means that the whole surface will be covered, but it is not an optimised radius since a smaller radius exists which also covers the whole of the surface of the polygon. A smaller radius will result in less overlapping of already covered areas.

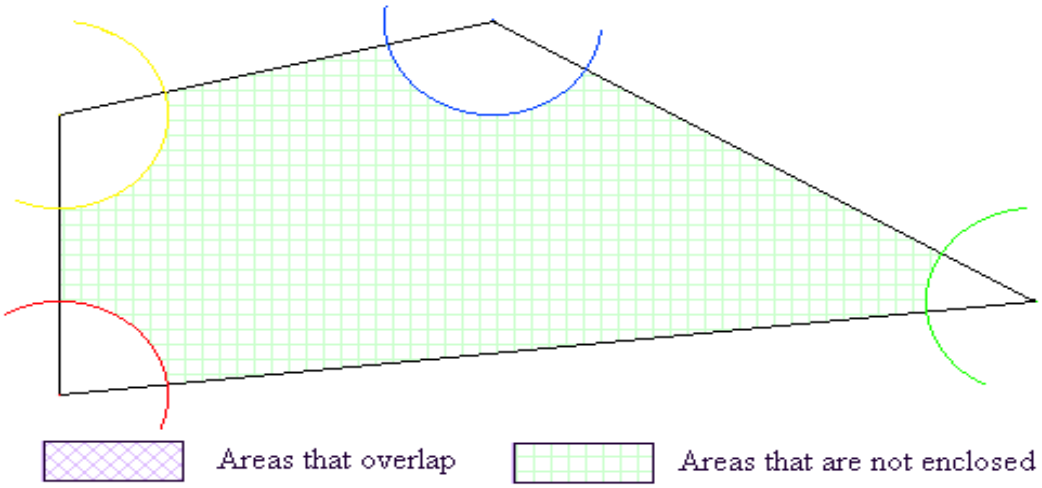

Figure 7.6 - Radius too small which leaves areas not enclosed.

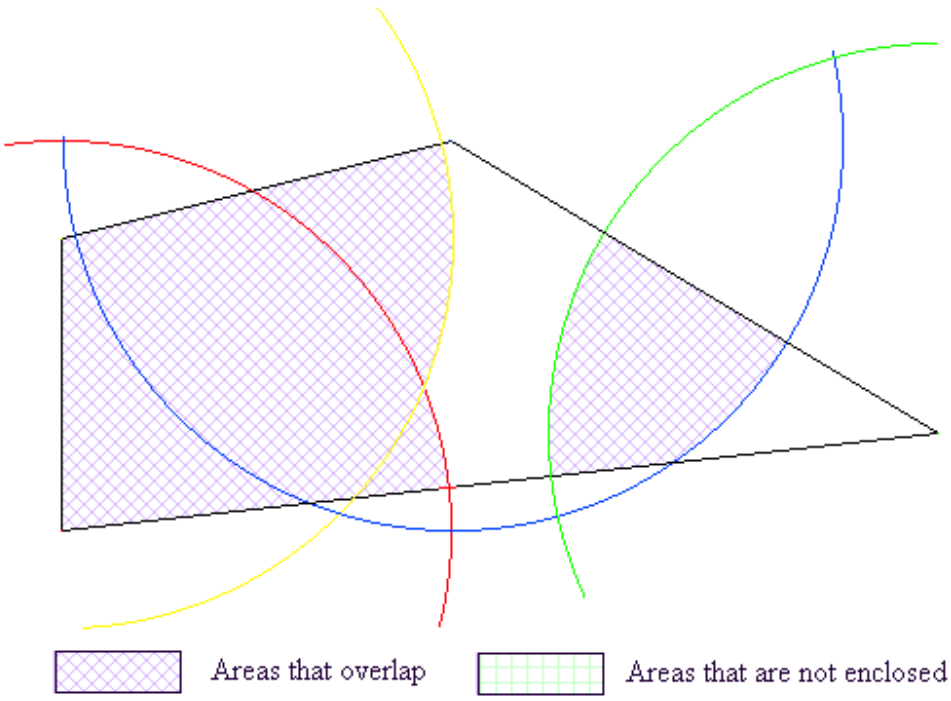

Figure 7.7 - Radius too big, which leaves large overlapping areas.

Two methods for calculating the warp detection distance were developed. At the time of writing this thesis, the first method, which is not the most optimised, was still implemented, but the second method will soon be implemented in place of the first one. These two methods will now be discussed.

The first method might not be the most optimised one, but the results are still satisfying. This method calculates a radius that is actually too big, but this is better than working with a radius that is too small. The following paragraphs will explain how this method of calculating each polygon's size works. Keep in mind that this system will always make use of polygons with four sides, therefore all the examples will make use of four-sided polygons.

Firstly, for each of the four lines between consecutive vertices, the middle point has to be calculated. For each of these middle points, distances to each vertex in the polygon must be calculated. The minimum radius is then calculated as the largest of these values divided by two. For example:

Figure 7.8 shows a four-sided polygon with the following vertices:
Vertex 1 (100,150), Vertex 2 (100,300), Vertex 3 (550,250) and Vertex 4 (300,100).

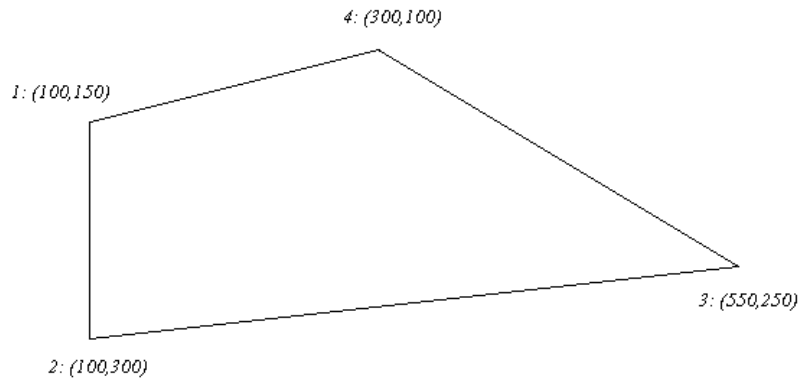

Figure 7.8 – A polygon with four vertices.

Figure 7.9 shows the middle points that were calculated:
Middle Point 1 between Vertex 1 and Vertex 2 (100,225)
Middle Point 2 between Vertex 2 and Vertex 3 (325,275)
Middle Point 3 between Vertex 3 and Vertex 4 (425,175)
Middle Point 4 between Vertex 4 and Vertex 1 (200,125)

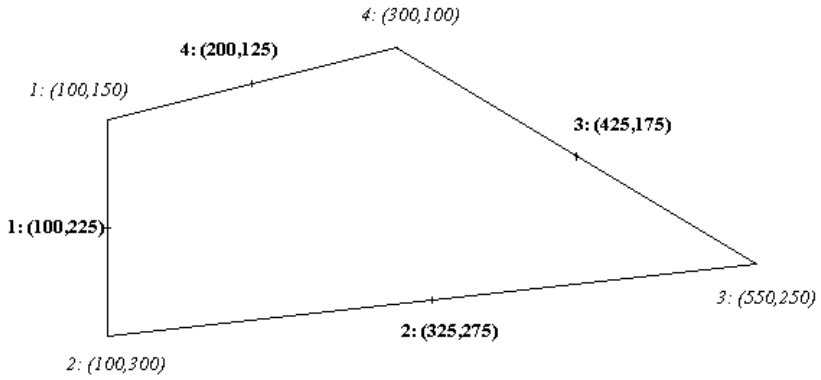

Figure 7.9 – Polygon with middle points indicated.

Figure 7.10 shows the distances from Middle Point 1 to each of the vertices of the polygon:

Distance from Middle Point 1 to Vertex 1 is: 75

Distance from Middle Point 1 to Vertex 2 is: 75

Distance from Middle Point 1 to Vertex 3 is: 450.69

Distance from Middle Point 1 to Vertex 4 is: 235.85

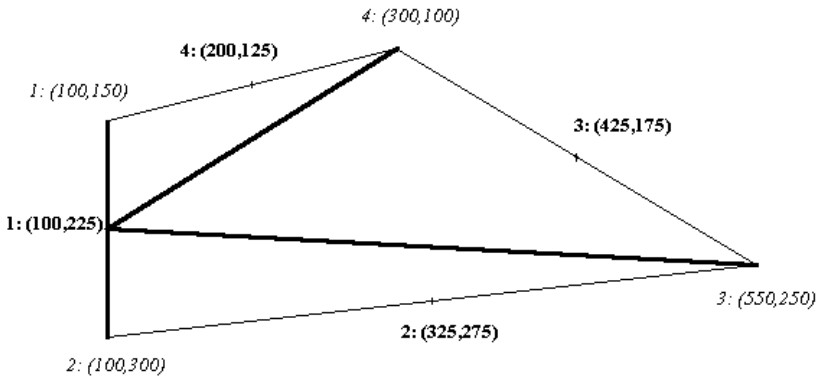

Figure 7.10 – Calculation of distances from Middle Point 1 to polygon vertices.

The same can be done for Middle Point 2, Middle Point 3 and Middle Point 4. The following results follow:

Distance from Middle Point 2 to Vertex 1 is: 257.39
Distance from Middle Point 2 to Vertex 2 is: 226.38
Distance from Middle Point 2 to Vertex 3 is: 226.38
Distance from Middle Point 2 to Vertex 4 is: 176.78

Distance from Middle Point 3 to Vertex 1 is: 325.96
Distance from Middle Point 3 to Vertex 2 is: 348.21
Distance from Middle Point 3 to Vertex 3 is: 145.77
Distance from Middle Point 3 to Vertex 4 is: 145.77

Distance from Middle Point 4 to Vertex 1 is: 103.01
Distance from Middle Point 4 to Vertex 2 is: 201.56
Distance from Middle Point 4 to Vertex 3 is: 371.65
Distance from Middle Point 4 to Vertex 4 is: 103.01

The largest of these values is the distance between middle point 1 and vertex 3, which is 450.69. The minimum radius is then calculated as 450.69 / 2 = 225.345.

Referring to Figure 7.11, it can clearly be seen that this method of calculating the size of each polygon is not an optimum solution since much of the areas defined by the calculated radius overlap. The red area in the figure gives an indication of how unoptimised this method is. The white areas in the figure show the areas that are covered but do not overlap. There are no green areas which means that the whole surface of the polygon is covered.

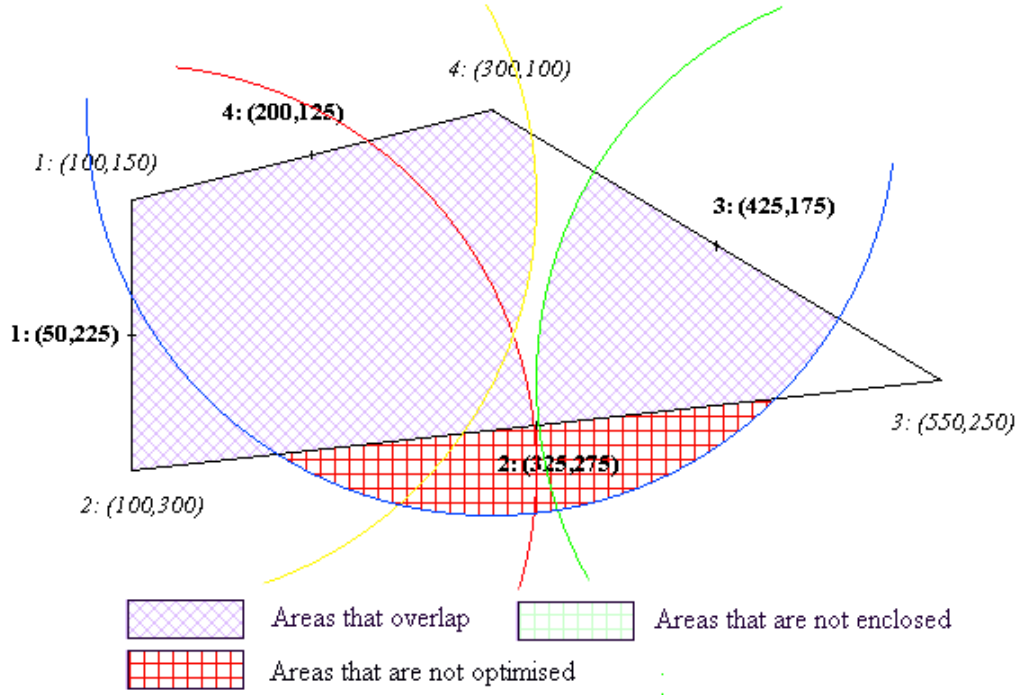

Figure 7.11 – Shows the minimum radius calculated for this polygon using the first method.

The pseudo code for calculating the warp detection distance is as follows:

```
1. Warp Detection Distance = 0
2. For all polygons in the model do:
 1. Get polygon vertex indices in vertex
 2. middle_point[0].x = vertex[0].x + (vertex[1].x – vertex[0].x)/2
 3. middle_point[0].y = vertex[0].y + (vertex[1].y – vertex[0].y)/2
 4. middle_point[0].z = vertex[0].z + (vertex[1].z – vertex[0].z)/2
 5. middle_point[1].x = vertex[1].x + (vertex[2].x – vertex[1].x)/2
 6. middle_point[1].y = vertex[1].y + (vertex[2].y – vertex[1].y)/2
 7. middle_point[1].z = vertex[1].z + (vertex[2].z – vertex[1].z)/2
 8. middle_point[2].x = vertex[2].x + (vertex[3].x – vertex[2].x)/2
 9. middle_point[2].y = vertex[2].y + (vertex[3].y – vertex[2].y)/2
 10. middle_point[2].z = vertex[2].z + (vertex[3].z – vertex[2].z)/2
 11. middle_point[3]].x = vertex[3].x + (vertex[0].x – vertex[3].x)/2
 12. middle_point[3].y = vertex[3].y + (vertex[0].y – vertex[3].y)/2
 13. middle_point[3].z = vertex[3].z + (vertex[0].z – vertex[3].z)/2
 14 for cnt = 0 to 3 do
  1. for dist_cnt = 0 to 3 do
   1. Dist = sqrt (     (middle_point[cnt].x-vertex[dist_cnt].x)² +
                        (middle_point[cnt].y-vertex[dist_cnt].y)² +
                        (middle_point[cnt].z-vertex[dist_cnt].z)² ) / 2
   2. if Dist > Warp Detection Distance
    1. Warp Detection Distance = Dist
```

Pseudo code line 2.1 retrieves the polygon's vertex indices in the list of vertices, not the actual vertices. Pseudo code lines 2.2 to 2.4 calculate Middle Point 1, lines 2.5 to 2.7 calculate Middle Point 2, lines 2.8 to 2.10 calculate Middle Point 3 and lines 2.11 to 2.13

calculate Middle Point 4. Lines 2.14.1 and 2.14.2 calculate distances from all the middle points to all the vertices to find a distance larger than the current warp detection distance.

The second method for calculating the warp detection radius calculates the optimum minimum radius, but it is not yet implemented. Referring to Figure 7.13, a point **P** has to be calculated so that the distance from each vertex in the polygon to point **P** is the same length. Point **P** can be called the centre of mass point. The distance from any vertex to point **P** can then be used as the size of the polygon.

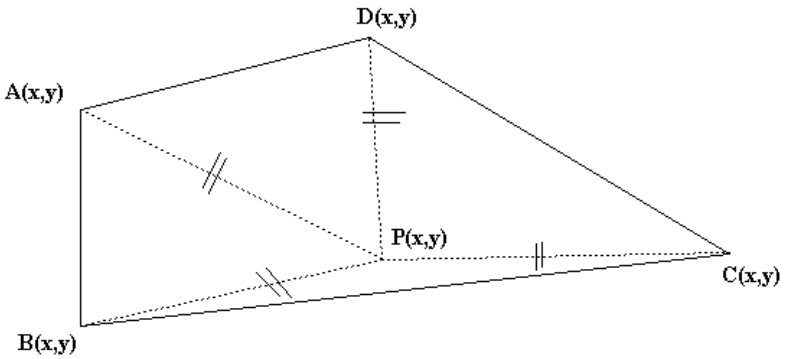

Figure 7.12 – Polygon with point **P** as the centre of mass point.

Therefore, using the distance formula, the following can be set up:

$$DistAP = \sqrt{(Px - Ax)^2 + (Py - Ay)^2} \tag{7-17}$$

$$DistBP = \sqrt{(Px - Bx)^2 + (Py - By)^2} \tag{7-18}$$

$$DistCP = \sqrt{(Px - Cx)^2 + (Py - Cy)^2} \tag{7-19}$$

$$DistDP = \sqrt{(Px - Dx)^2 + (Py - Dy)^2} \tag{7-20}$$

Since these distances have to be the same, the following six equations follow:

DistAP = DistBP:

$$\sqrt{(Px-Ax)^2+(Py-Ay)^2}=\sqrt{(Px-Bx)^2+(Py-By)^2} \quad (7\text{-}21)$$

DistAP = DistCP:

$$\sqrt{(Px-Ax)^2+(Py-Ay)^2}=\sqrt{(Px-Cx)^2+(Py-Cy)^2} \quad (7\text{-}22)$$

DistAP = DistDP:

$$\sqrt{(Px-Ax)^2+(Py-Ay)^2}=\sqrt{(Px-Dx)^2+(Py-Dy)^2} \quad (7\text{-}23)$$

DistBP = DistCP:

$$\sqrt{(Px-Bx)^2+(Py-By)^2}=\sqrt{(Px-Cx)^2+(Py-Cy)^2} \quad (7\text{-}24)$$

DistBP = DistDP:

$$\sqrt{(Px-Bx)^2+(Py-By)^2}=\sqrt{(Px-Dx)^2+(Py-Dy)^2} \quad (7\text{-}25)$$

DistCP = DistDP:

$$\sqrt{(Px-Cx)^2+(Py-Cy)^2}=\sqrt{(Px-Dx)^2+(Py-Dy)^2} \quad (7\text{-}26)$$

The problem is that there are six equations but only two unknown variables, namely **P**x and **P**y, since points **A**, **B**, **C** and **D** are known. This means that a precise solution can never be found, only an approximated solution. In other words, instead of finding a single *point* for a solution, an *area* will be found for an approximated solution. In 3-D a *volume* will be found as an approximated solution. Any point in the area or volume can be used for an approximated solution. This method can be used recursively until a satisfying value is found.

Each of the six equations can also be written in a different format as the equation for a line:

$$y = ax + b \tag{7-27}$$

Using line 1 and line 2 to get an intersection point, then line 2 and line 3 for another intersection point, and so on, five intersection points can be calculated. These five points form a polygon in which the optimum minimum radius is confined. As a first level search for finding the optimum minimum radius, these five points can be used to calculate an average point. This point is then taken as the centre of mass point. This method can also be applied recursively on the five sided polygon defined by the intersection points if a satisfying value were not found.

To compare the two methods, the same example will be worked through using the second method.

The six lines obtained from equations (7-21) to (7-26) are:

Line 1: $y = (0)x + 255$ (7-28)

Line 2: $y = (-4.5)x + 1662.5$ (7-29)

Line 3: $y = (4)x – 675$ (7-30)

Line 4: $y = 9x – 2650$ (7-31)

Line 5: $y = 1x + 0$ (7-32)

Line 6: $y = 1.66667 + 883.33333$ (7-33)

The intersection points are the following:

Line 1 with Line 2: $x = 319.44444$
$y = 225$

Line 2 with Line 3: $x = 275$
$y = 425$

Line 3 with Line 4: x = 395
y = 905

Line 4 with Line 5: x = 331.25
y = 331.25

Line 5 with Line 6: x = 331.25
y = 331.25

Therefore the average point is: x = 330.38889
y = 443.5

The centre of mass point is therefore defined as: (330.38889,443.5)

The distance from each vertex in the polygon to this centre of mass point must be calculated:

Distance to vertex 1: 373.12369
Distance to vertex 2: 271.42456
Distance to vertex 3: 292.69658
Distance to vertex 4: 344.84161

The longest distance is the distance from the centre of mass point to vertex 1. Therefore the optimum minimum radius is: 373.12369 / 2.0 = 186.56185. Comparing this value with the first method's value of 225.345, it can clearly be seen that the radius calculated using the second method is more optimal.

Referring to Figure 7.13, it can clearly be seen that this method of calculating the size of each polygon is an optimum solution since much less of the areas defined by the calculated radius overlap and the amount of red area in the figure indicates how much this method is optimised. Using the second method recursively will reduce the red area even further. Theoretically speaking, the red area will never vanish but it will strive to with each

recursion. The white areas in the figure show the areas that are covered but do not overlap. There are no green areas which means that the whole surface of the polygon is covered.

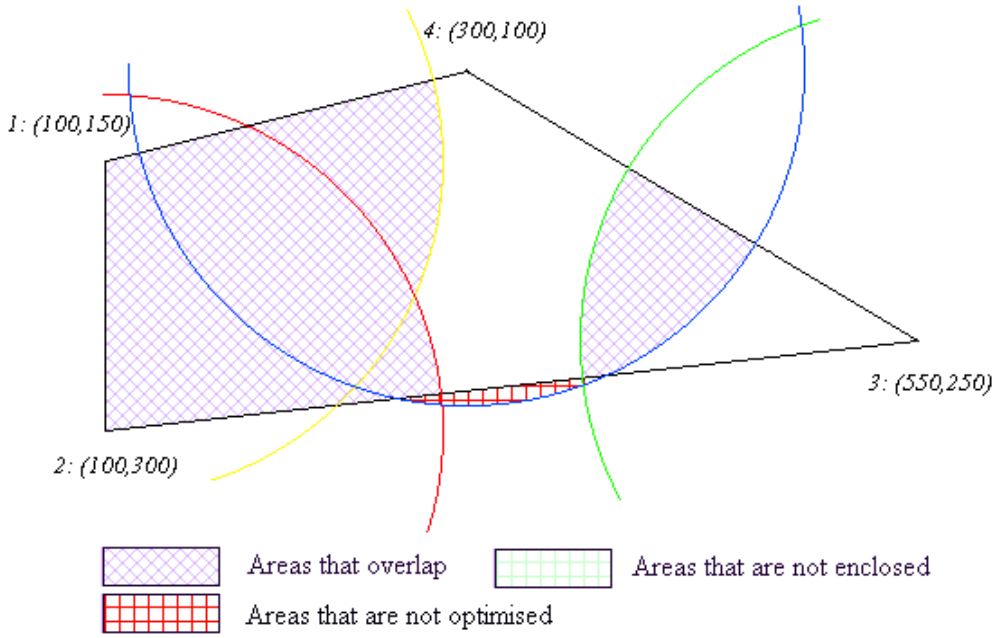

Figure 7.13 – Show2 the minimum radius calculated for this polygon using the second method.

7.4.2 Computer Graphic User Interface

The *computer graphic user interface* of the system is the normal computer interface that the user will work with. Refer to section 7.2 for a detailed description of the system's user interfaces. Input devices for the computer graphic user interface are a mouse, keyboard and the 3-D tracker. The computer screen serves as an output device. The system's main window with a menu is displayed on the computer screen. The main window shows the generated image as if a real procedure were done using the gastroscope. It also has a status line at the bottom, which shows what is currently being done or what the user is supposed to do. For example, "Loading Identification Data" or "Click on a condition". Using the menu, the user can perform certain tasks for training purposes, like setting up scenarios with abnormal conditions. Some of the menu items are dynamic items, like Preparatory Procedures, Images of Conditions and Video Clips of Real Procedures. The reason for this is that in future the simulator could be used for simulations of other organs, like the lungs.

For each organ, these menu items will have different data, so it will be necessary to change the items dynamically. At the moment the simulator only works using a 3-D tracking device attached to a gastroscope, but in future the user will also be able to navigate inside the VR model using the mouse and buttons. It is unfortunately very difficult to navigate in 3-D space using a normal 2-D mouse, but fixed paths might be set so that a user can use the mouse to move along the fixed path for exploring the inside of the organ. All windows and dialog boxes were created using low-level Windows 95 API (Application Programming Interface) function calls. The following section will discuss the 3-D tracking device used in this system in detail.

7.4.2.1 3-D Tracker

The 3-D tracking device used with this system is a six-degrees-of-freedom, electromagnetic device, manufactured by Polhemus. The tracking device's transmitter is placed at a specific place near the physical model of the stomach. A plastic fitting was moulded over the receiver so that the receiver can easily fit onto the front tip of a gastroscope. Plate 2.4 illustrates the plastic fitting moulded over the receiver. As the gastroscope is then slid into the physical model, the tracking device can determine exactly where the tip of the gastroscope is as well as the orientation of the tip of the gastroscope. The tracking device is initialised (in the initialisation module) to return the tracked position in the measure of centimetres. The Polhemus tracking device can also return the position in the measure of inches.

With each frame that is rendered, the computer graphic user interface reads the position (x, y, z) and orientation (yaw, pitch, roll) from the 3-D tracking device. This orientation and position is applied to the virtual camera in the scene that correlates to the tip of the physical gastroscope. The position will have to be translated and scaled because the VR model is normalised to fit in a unit cube. Render engines normally have built-in vector and matrix functions for transformation of VR models. Renderware's matrix and vector functions were used. Transformation matrices are used to scale, move or change the rotation of a VR model.

The orientation of the viewpoint of the virtual camera correlates with the orientation of the tip of the gastroscope. The 3-D position measured by the tracking device is scaled and translated so that the position of the viewpoint of the virtual camera correlates with the position of the tip of the gastroscope inside the physical model. The scaling and translation factors have been calculated during the registration of the VR model with the physical model. The transformation calculations are done in the computer graphics user interface each time a frame needs to be rendered.

Pseudo code for setting up the transformation matrix follows:

```
1. Read data from 3-D tracking device XPos, YPos, ZPos, XRot, YRot, Zrot
2. Initialise Matrix as a unit matrix
3. RotateMatrix (Matrix, 0,1,0, YRot , REPLACE)
4. RotateMatrix (Matrix, 1,0,0, XRot , PRECONCATENATE)
5. RotateMatrix (Matrix, 0,0,1, ZRot , PRECONCATENATE)
6. TranslateMatrix (Matrix,-(XPos+RegisterX)/STOMACH_SCALE_X,
                           -(YPos+RegisterY)/STOMACH_SCALE_Y,
                           -(ZPos+RegisterZ)/STOMACH_SCALE_Z,
                                             POSTCONCATENATE)
```

Pseudo code lines 2 to 5 set up the matrix for the virtual camera's orientation and line 6 sets up the matrix for the virtual camera's position. RotateMatrix() is an API function of Renderware. The first parameter is the matrix on which the rotation will be applied. The next three parameters are coordinates of a 3-D vector around which the rotation should be done. The fifth parameter indicates the degrees of rotation around the specified vector. The last parameter can either be "REPLACE", "PRECONCATENATE" or "POSTCONCATENATE". This parameter indicates the order in which the matrix should be concatenated to the existing one. "REPLACE" means that the matrix is replaced by the function's result. "PRECONCATENATE" means that the resulting matrix is multiplied by the existing matrix in such a manner that the resulting matrix is pre-concatenated to the existing matrix. "POSTCONTATENATE" means to apply the resulting matrix to the existing one in such a manner that the resulting matrix is post-concatenated to the existing matrix. Line 8 moves the tip of the virtual camera to the correct position. Three values for each coordinate are involved, the read values from the 3-D tracking device, the registration values for each coordinate and the scaling factor calculated during the normalisation of the VR model.

7.4.3 Original VR Model

Referring to Figure 7.1, it can be seen that there are two VR models used in the system. This computer data set is called the *original VR model* because it is the VR model generated by digitising a physical model and registering it. The original VR model is normalised and smoothed during initialisation of the system.

The reason why two VR models are needed is because of the dynamic changes to the VR model while warping or transforming the VR model. These dynamic or real-time changes are necessary when the tip of the gastroscope almost touches the inside of the VR model to warp the model so that the virtual camera cannot penetrate the VR model and show an image outside the VR model. Plates 7.8 and 7.9 illustrate how the system's main window and camera orientation windows look like with warping and without. To be able to restore the warped model to its original form, a copy of the original VR model must be kept. This original VR model's information is also used when adding 3-D conditions. See section 7.4.6.1 for more information on the addition of 3-D conditions to the VR model.

7.4.4 Real-Time Transformation of Original VR Model

With each frame that needs to be rendered, this module uses the input from the *computer graphic user interface* to calculate where the tip of the gastroscope is in the VR model. The computer graphic user interface calculates a transformation matrix with each frame that needs to be rendered. This transformation matrix can be used to set the virtual camera's position and orientation. If the tip is pressing against the side of the VR model, the VR model's shape must be deformed so that the tip cannot penetrate the VR model. To deform the shape of the VR model some of the VR model's vertices have to be warped away from the tip of the gastroscope. When the VR model is deformed near an area where 3-D conditions were placed, the 3-D conditions also need to be warped. To warp the 3-D conditions, the module needs to communicate with the *3-D condition database* so that information of the original 3-D condition can be found. During some navigational tasks the anatomic regions must be shown to the user by changing the colours of the different regions. To be able to show certain regions in the stomach, the module needs to be able to communicate with the *region database*.

The following two sections will discuss collision detection and warping. For the VR model's shape to be deformed in real-time, the system first needs to know if and where the tip of the gastroscope almost touches the VR model. If the tip almost touches the VR model, then the vertices at that area must be warped way from the tip of the gastroscope.

7.4.4.1 Collision Detection

It is very important that the VR model is not just stationary, but a dynamic model that will deform or bend as the gastroscope almost touches the inner sides of the VR model. The most important aspect of this warping process is collision detection. Collision detection is the process of determining whether two objects have collided. In this case, collision between the tip of the gastroscope and the VR model has to be detected.

Several techniques are used for collision detection. The mini-max testing method can be seen as a first level filter to reduce calculations. This is a method of testing collision between two 2-D shapes by comparing the shapes' minimum and maximum x- and y-extends [VIN95]. Figure 7.14 shows how the mini-max testing for 2-D shapes works. A boundary square is calculated for each shape. The minimum and maximum values of the boundary square can then be used to check if two shapes' boundary squares overlap. When the boundary squares overlap, a *possible* collision is detected, otherwise a collision is *not possible*. If there is a possibility of collision, each vertex must be tested against each surface of the other object. This will be explained later in this section. The advantage of using this method is that it requires very few calculations.

Another method is to use boundary circles around each shape. The radius of each boundary circle is then used to check if two shapes overlap. Figure 7.15 shows how boundary circles can be used for collision detection of 2-D shapes.

By adding a third axis to a boundary square or circle, 3-D objects can also be checked in the same way. A boundary box is an extension of a boundary rectangle and can be used with the mini-max method to check for collisions between 3-D shapes. This basically involves adding the minimum and maximum z-extend of the 3-D shape. A boundary sphere is an extension of a boundary circle. The boundary sphere's radius must be adjusted

to enclose all the vertices. To check if two objects have collided, the distance between the centre points of each object is calculated and compared to the radii of the boundary spheres. If the distance is greater than the sum of the two radii, then the two objects cannot collide, otherwise the objects *could* have collided [VIN95].

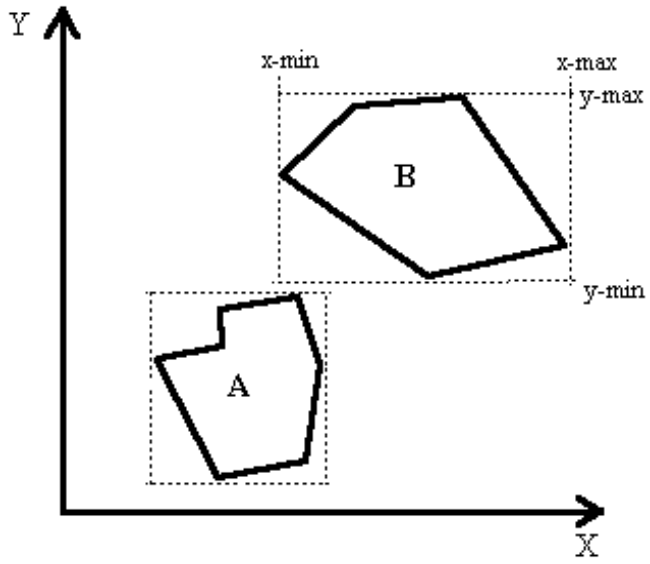

Figure 7.14 – Shows how boundary rectangles can be used for the mini-max collision detection.

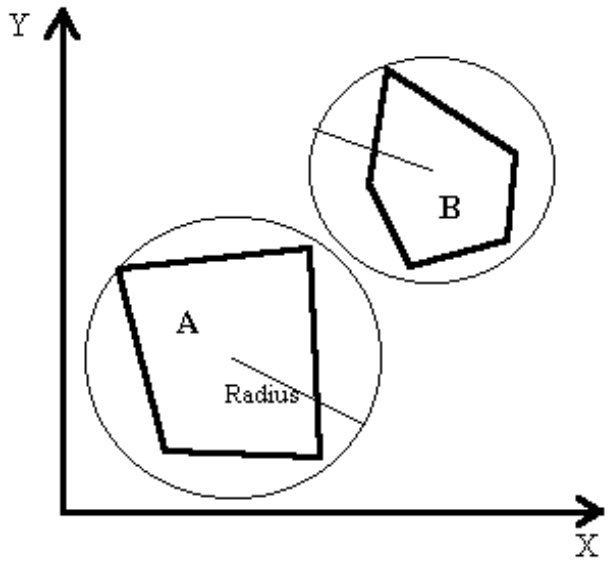

Figure 7.15 – Shows how boundary circles can be used for collision detection.

If a possibility of a collision exists, each vertex of one object must be tested against each surface of the other object for a collision. Referring to Figure 7.16, to know if an edge formed by two vertices P*a* and P*b* intersects a polygon, the following calculations can be done:

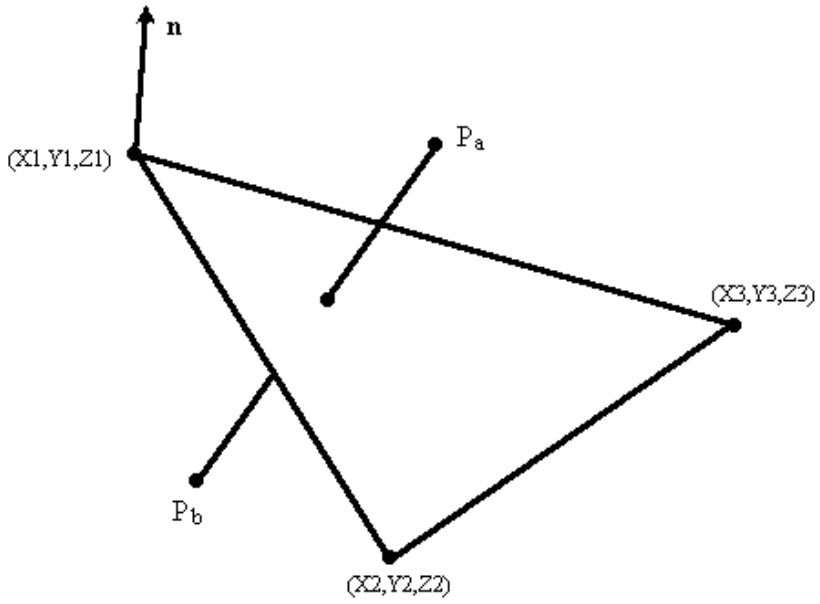

Figure 7.16 – Shows a polygon and two points P*a* and P*b* which could be vertices from another polygon forming and edge.

If **n** = [a, b, c] is the normal vector of the polygon, then the polygon's surface can be given by:

$$ax + by + cz + d = 0 \qquad (7\text{-}34)$$

"d" can be determined by substituting (x_1, y_1, z_1) in equation (7-34).

$$d = -(ax_1 + by_1 + cz_1) \qquad (7\text{-}35)$$

By substituting "d" into equation (7-34), equation (7-36) can be written for any point P(x, y, z) on the surface.

$$ax + by + cz – (ax_1 + by_1 + cz_1) = 0 \quad (7\text{-}36)$$

This can also be written as:

$$a(x – x_1) + b(y – y_1) + c(z – z_1) = 0 \quad (7\text{-}37)$$

If P(x, y, z) is *not on* the surface, but in the half-space containing **n**, the left hand side of equation (7-37) will be positive. If P is located on the opposite half-space, the expression will be negative. In Figure 7.16 P*a* is in the positive half-space and P*b* is in the negative half-space, which therefore means that the edge formed by vertex P*a* and P*b* intersects the polygon [VIN95].

In this system, the tip of the gastroscope must be checked for collision with the VR model. The tip of the gastroscope will mostly be inside the VR model, therefore the above method will not work because a collision will always be detected between the VR model and the tip of the gastroscope while the gastroscope is inside the VR model. The following method was therefore developed:

The tip of the gastroscope must be checked for collision with each polygon of the VR model as if each polygon were a separate object. If a collision is detected, the appropriate polygon(s) must be warped outwards. If all the polygons in the VR model have to be checked for collision, the process will be very slow. All the polygons do not need to be checked, only those that are actually facing the tip of the gastroscope. The following paragraph will explain how to calculate which polygons are facing the tip of the gastroscope using the normal vectors of each polygon. If a polygon's normal vector faces the tip of the gastroscope, it must be checked for collision with the tip of the gastroscope, otherwise no other calculations need do be done with that polygon.

For each polygon in the VR model the following has to be done with each frame that is to be rendered. Firstly, the normal vector of each polygon and the unit direction vector of the tip of the gastroscope have to be determined. The normal vectors of each polygon are calculated during the initialisation process and the unit direction vector of the tip of the gastroscope is obtained from the 3-D tracking device. To determine if the tip of the

gastroscope is facing the polygon or not, the inner product or dot product can be used to calculate the angle between the polygon's normal vector and the unit direction vector of the tip of the gastroscope.

The inner or dot product of two vectors **a** and **b** is defined as:

$$\mathbf{a} \bullet \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos(\alpha) \qquad \text{if } \mathbf{a} \neq 0 \text{ and } \mathbf{b} \neq 0 \qquad (7\text{-}38)$$

$$\mathbf{a} \bullet \mathbf{b} = 0 \qquad \text{if } \mathbf{a} = 0 \text{ and } \mathbf{b} = 0 \qquad (7\text{-}39)$$

where α is the angle between **a** and **b**.

Using the ArcCos function, the angle between the vectors can be calculated. If the dot product is –1.0, the angle between the vectors is 180 degrees. If the dot product is 0.0, the angle is 90 degrees [SAL90]. The system does not have to calculate the ArcCos function each time as it uses the result of the dot product. Knowing the above, the system calculates the dot product, and if the result is between –1.0 and 0.0 the polygon is facing the tip of the gastroscope and needs to be checked for collision. In Figure 7.17 the first situation is of a polygon facing the tip of the gastroscope. Although not directly, this polygon must still be examined for collision detection. The second situation is of a polygon directly facing the tip of the gastroscope. The last situation is of a polygon not facing the tip of the gastroscope and will therefore not be examined for collision detection.

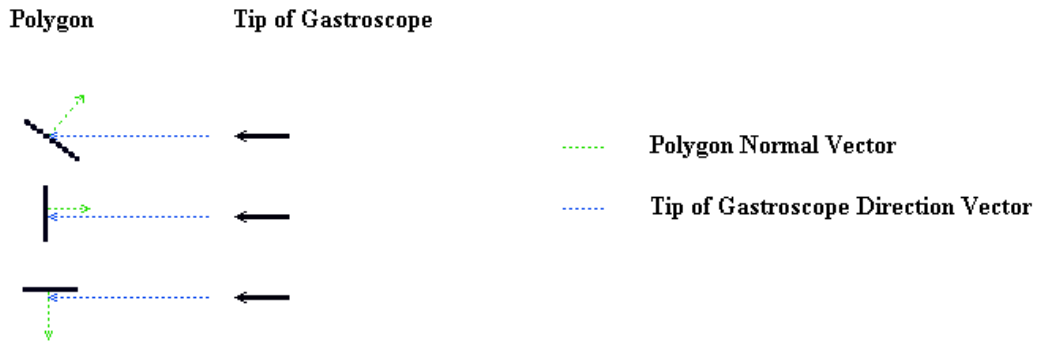

Figure 7.17 – Three situations showing the angle between the polygon normal vector and the direction vector of the tip of the gastroscope.

Theoretically speaking only values between 180 degrees and 90 degrees have to be checked. However, because the typical gastroscope uses a large field of view, for example 120 degrees, more than just the polygons facing the gastroscope have to be checked. The system uses the dot product results between –1.0 and 0.20, which corresponds to 180 degrees to 78.5 degrees. The number 0.20 was chosen purely from a trial and error experiment. This could be an aspect of the system to examine if the system needs to be optimised for speed.

The method used in this system for collision detection is very simple. The closest polygon facing the tip of the gastroscope is determined by calculating the distance from the tip of the gastroscope to each polygon facing the tip of the gastroscope. A distance, the warp detection distance, is calculated in the initialisation module. This distance is used to determine if a polygon that is facing the tip of the gastroscope is too close to the tip of the gastroscope and therefore must be warped away from the tip of the gastroscope. For each polygon facing the tip of the gastroscope, the distance from the tip of the gastroscope to the centre of each polygon must be calculated, using:

$$Dist = \sqrt{(Centre.x - Tip.x)^2 + (Centre.y - Tip.y)^2 + (Centre.z - Tip.z)^2} \qquad (7\text{-}40)$$

If the distance *Dist* of the closest polygon facing the tip of the gastroscope is less than the warp detection distance, this polygon must be warped. Warping only the polygon where the collision has been detected will result in an unnatural shape. The VR model will look normal, but with one polygon warped away and giving the VR model an uneven look. Therefore the polygons in the neighbourhood of this polygon should also be warped, but not as far. A distance called the warp radius is defined in the system. All polygons that are within the warp radius are put in a list that will be used during warping. The warp radius can be seen as the radius of a sphere around the position of the tip of the gastroscope with the position of the tip as the sphere's centre. Any polygons facing the tip of the gastroscope *and* within this sphere will also be checked for warping.

The pseudo code for the collision detection is as follows:

```
1. Min_Distance = 32000       {ensures the first test value is minimum}
2. ListPos = 0
3. Get Camera look at direction in camera_dir
4. Normalize camera_dir
5. Get Camera position in camera_pos
6. For all polygons in model do:
 1. Get polygon normal vector
 2. Dot = DotProduct (polygon normal, camera_dir)
 3. if Dot < 0.20   {then camera faces polygon}
  1. Get polygon center
  2. Distance = sqrt ( (camera_pos.x-center.x)² +
                       (camera_pos.y-center.y)² +
                       (camera_pos.z-center.z)² )
  3. if Distance < Min_Distance
   1. Min_Distance = Distance
   2. Closest Polygon = current polygon
  4. if Distance <= WarpRadius  {then add polygon to warp list}
   1. PolygonWarpList[ListPos] = current polygon
   2. ListPos = ListPos + 1
```

Pseudo code line 3 gets the direction in which the tip of the gastroscope, or the virtual camera, looks at as a 3-D vector “camera_dir”. This vector should be normalised to do calculations such as dot products, therefore line 4 normalises it. Pseudo code lines 6.1 to 6.3 first test if the current polygon is facing the tip of the gastroscope. As explained earlier, the value 0.20 is used because of the wide-angle lenses used in gastroscopes. If a polygon is facing the tip of the gastroscope, the distance to the polygon centre is calculated in line 6.3.2. Line 6.3.3.1 and line 6.3.3.2 set the minimum distance of the closest polygon to the tip of the gastroscope. Line 6.3.4 checks if the current polygon is within the warp radius and if it is, adds it to a list of polygons to be warped in the warp procedure.

This method of collision detection is different than the mini-max bounding box method described earlier, since this method tests collisions between a point (with a direction vector) and other polygons. The mini-max bounding box test method cannot be used in this case since it would be meaningless to have minimum and maximum values for a single point. The developed method rather resembles that of the bounding sphere method except that the point also has a direction that can be used to optimise the collision detection.

This section explained how it is determined which polygons are too close to the tip of the gastroscope. The following section will explain how these polygons are warped away from the tip of the gastroscope and how it is determined how far and in which direction each polygon should be warped.

7.4.4.2 Warping

Warping is the real-time transformation of a 3-D object's vertices so that the 3-D object's shape is not rigid, but can be changed or deformed dynamically. When a force is applied to the object, the object's shape changes and as soon as the force is taken away, the 3-D object changes to its original shape.

This system needs to implement warping so that whenever the tip of the gastroscope almost touches the inside of the VR model, the VR model will be deformed so that the tip of the gastroscope can never penetrate the VR model. Using a rigid physical model and given that the VR model could be registered a 100% precisely with the physical model, it should not be necessary to implement warping. The problem is that it is very difficult to register the VR model precisely with the physical model. Future physical models will also be made of transparent rubbery material, which will require a VR model that can be warped[8]. If the tip of the gastroscope were to penetrate the VR model and move outside the VR model, the render engine will render an unrealistic image that shows an orange background colour. See Plate 7.8 and Plate 7.9 for examples of the system with and without warping. Plate 7.8 shows the system's main window and Plate 7.9 shows the system's orientation window. The following paragraphs will discuss the warping methods developed during the development of this system.

There are two main problems to deforming the VR model in this system. Firstly the vertices of the VR model must be warped in the direction that a force is applied and secondly, the vertices of the VR model must be warped back to its original positions when the force is taken away.

[8] At the time of this writing the current system is using a silicon model. This was, however, regarded as beyond the scope of this study and is therefore not included in this thesis.

Implementing the deforming of the VR model when a force is applied will be explained in detail in the following paragraphs. Deforming the shape of the VR model when the force is taken away can be done by resetting the VR model to its original shape each time before it is supposed to be warped. This means that the model will be deformed less with each frame that is rendered as the force is taken away. If the force is applied again, the model will also be reset to its original form and then deformed more with each frame rendered. The rest of this section will explain how warping was implemented in this system.

Firstly, it should be determined *if* the VR model should be warped and secondly *where* it should be warped. The collision detection routine will determine if the tip of the gastroscope is too close to the VR model. If the collision detection routine has determined that the tip of the gastroscope was not too close to the VR model, then no warping is needed. Otherwise a list of polygons that are too close to the tip of the gastroscope will be set up by the collision detection routine. These polygons are *where* the VR model must be warped. The polygons in this list must be warped away from the tip of the gastroscope so that the tip of the gastroscope is not too close to any of the polygons anymore.

Figure 7.18 and Figure 7.19 show situations where the distance from the tip of the gastroscope to the original (not warped) vertex of a polygon is greater than the warp detection distance and therefore too far away to be warped.

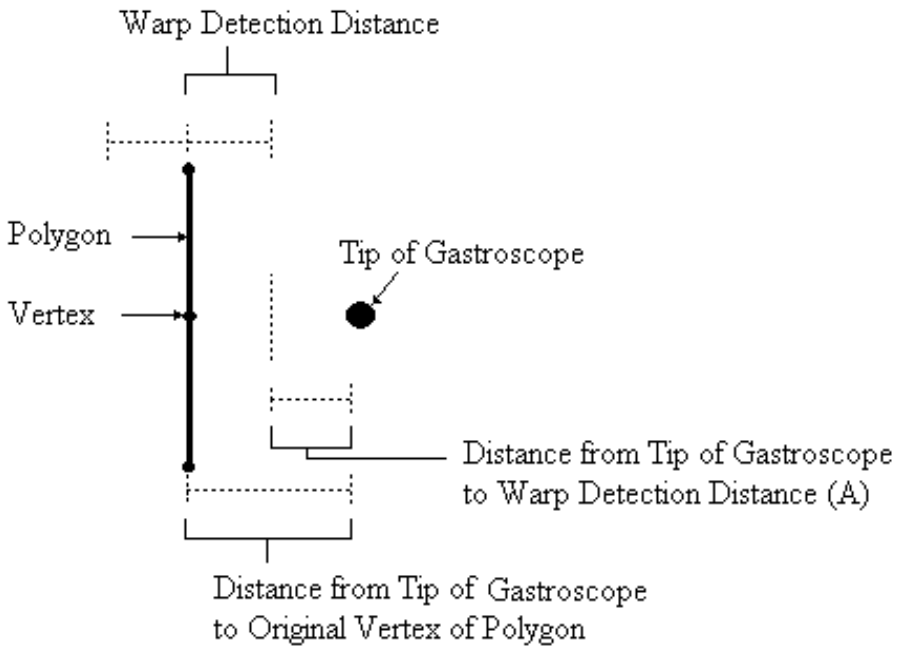

Figure 7.18 – Shows a situation where the tip of the gastroscope is too far away from the original (not warped) vertex of a polygon to be warped.

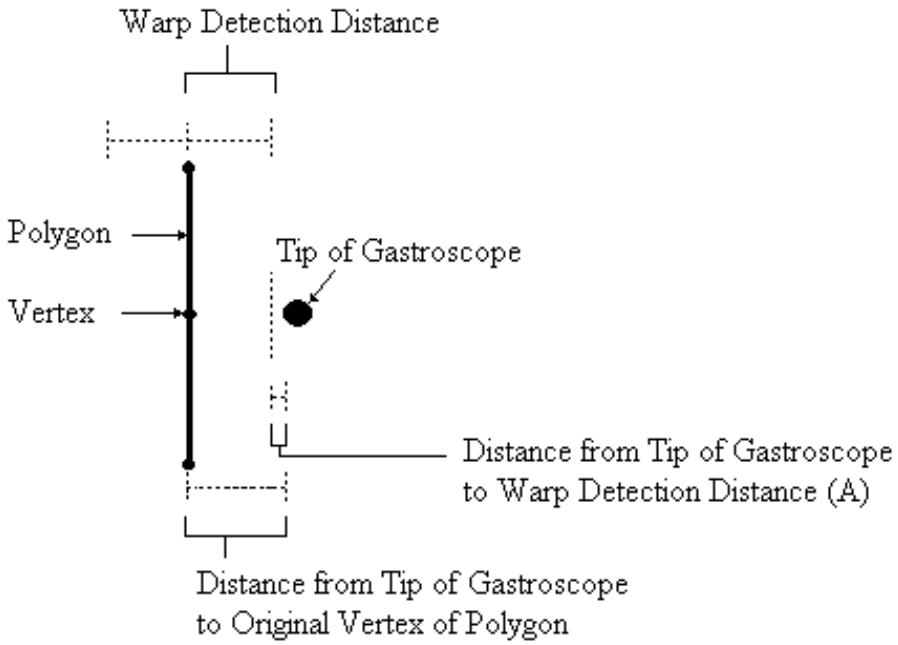

Figure 7.19 – Shows a situation where the tip of the gastroscope is too far away from the original (not warped) vertex of a polygon to be warped.

Figure 7.20 illustrates what happens with polygons that should be warped. To warp the polygon away from the tip of the gastroscope, each vertex in the polygon has to be moved a certain *distance* in a certain *direction*, so that the distance between the tip of the gastroscope and the vertex is equal to or greater than the warp detection distance. In Figure 7.20 the direction in which the vertex will be warped is the same direction of the "incoming" tip of the gastroscope. Later in this section it will be explained exactly how the direction in which each vertex is warped is calculated. Figure 7.20 clearly shows that the distance from the tip of the gastroscope to the original (not warped) vertex is less than the warp detection distance and should therefore be warped by distance A. Note that the tip of the gastroscope is still in front of the original vertex.

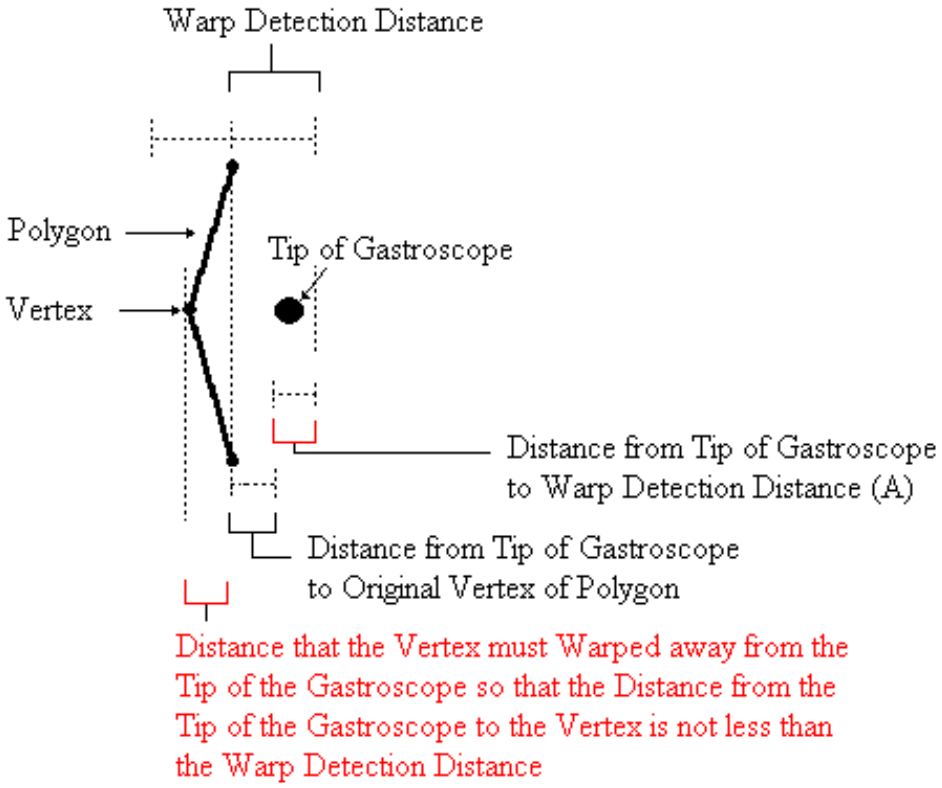

Figure 7.20 – Illustrates what happens with polygons that should be warped. In this case the tip of the gastroscope is still in front of the original vertex.

Figure 7.21 also shows a situation where the vertex of a polygon has to be warped away from the tip of the gastroscope, but in this case the tip of the gastroscope is beyond the original vertex. Note that distance A, the distance that the vertex should be warped, is greater than in Figure 7.20.

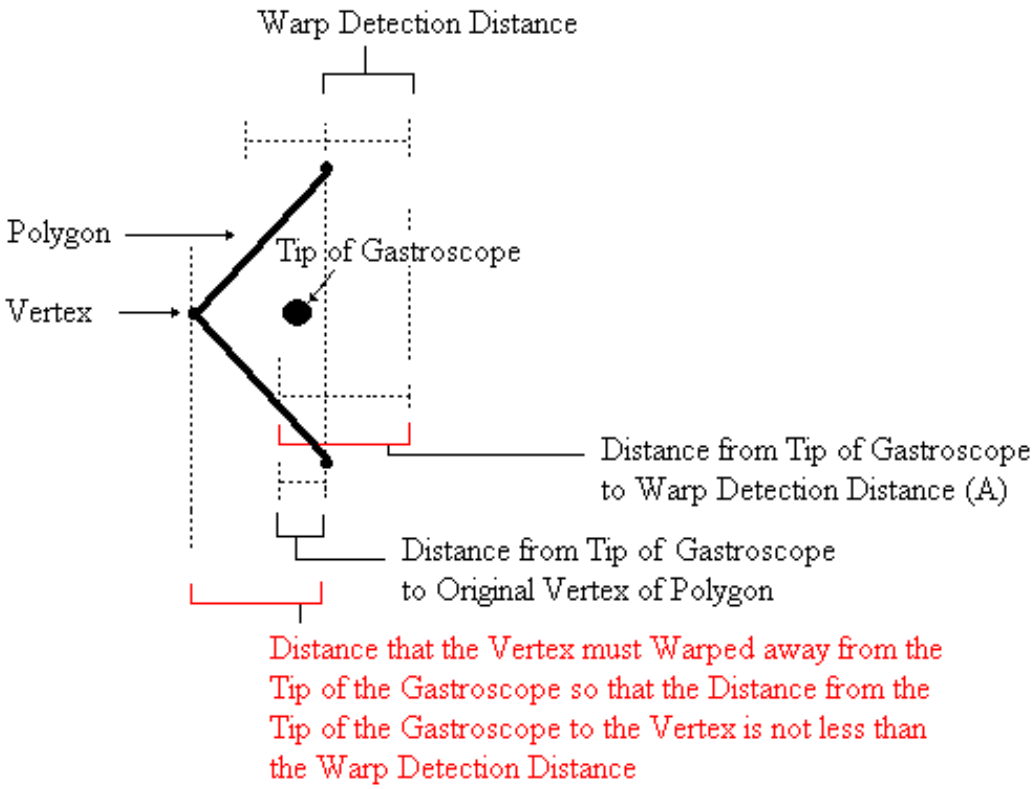

Figure 7.21 – Illustrates what happens to polygons that should be warped. In this case the tip of the gastroscope is beyond the original vertex.

The following paragraphs will explain in detail how this system's warping routine works. Some aspects that will be explained, are how far and in which direction each vertex of a polygon should be warped if it is in the list of polygons to be warped.

When the tip of the gastroscope gets too close to the inside of the VR model, the polygons that need to be warped have to be warped away from the tip of the gastroscope in a certain *direction* and by a certain distance in such a manner that it is not too close to the VR model anymore. Two different methods for choosing a direction were experimented with. The first method uses the direction of the tip of the gastroscope to warp polygons away from the tip of the gastroscope. This will warp all polygons in the *same* direction, directly away from the tip of the gastroscope. The second method uses each polygon's normal vector for a direction vector along which the polygon must be warped. This means that each polygon could be warped in a *different* direction depending on their normal vectors. If the side of the tip of the gastroscope gets too close to the inside of the VR model then warping also needs to be done. The first method will warp the polygons in the forward

direction of the tip of the gastroscope, which in some cases do not give very realistic results. Using the second method works very well since the deforming of the shape of the VR model is much more even than that of the first method. Figure 7.22 shows a situation where both approaches will work and Figure 7.23 shows a situation where using the second method works much better.

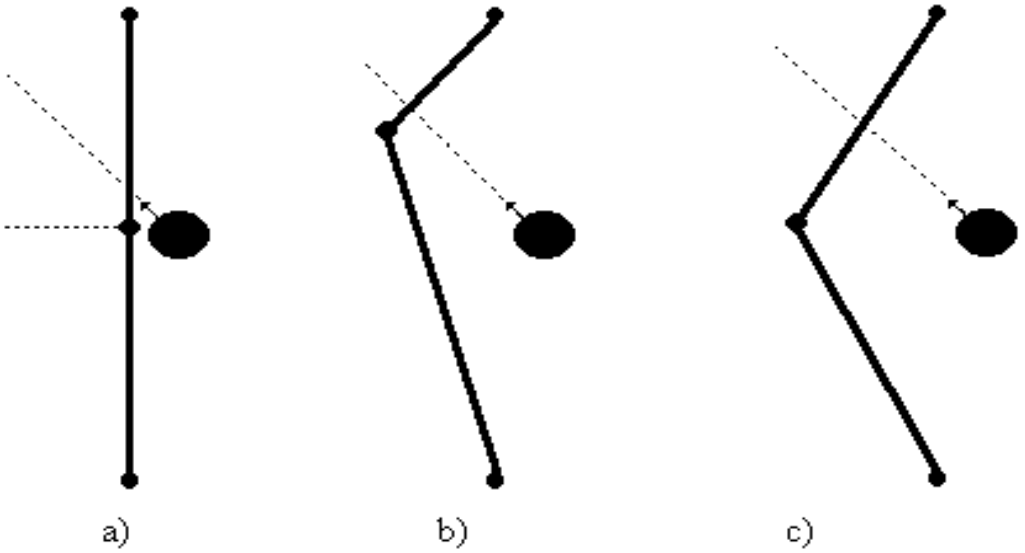

Figure 7.22 – Shows two different approaches to choosing a direction in which a vertex should be warped. Figure a) shows the tip of the gastroscope close to a vertex. The direction in which the tip of the gastroscope is pointed and the normal vector of the polygon are also shown. Figure b) shows how the vertex will be warped if it is warped in the direction of the tip of the gastroscope. Figure c) shows how the vertex will be warped using the normal vector of the polygon.

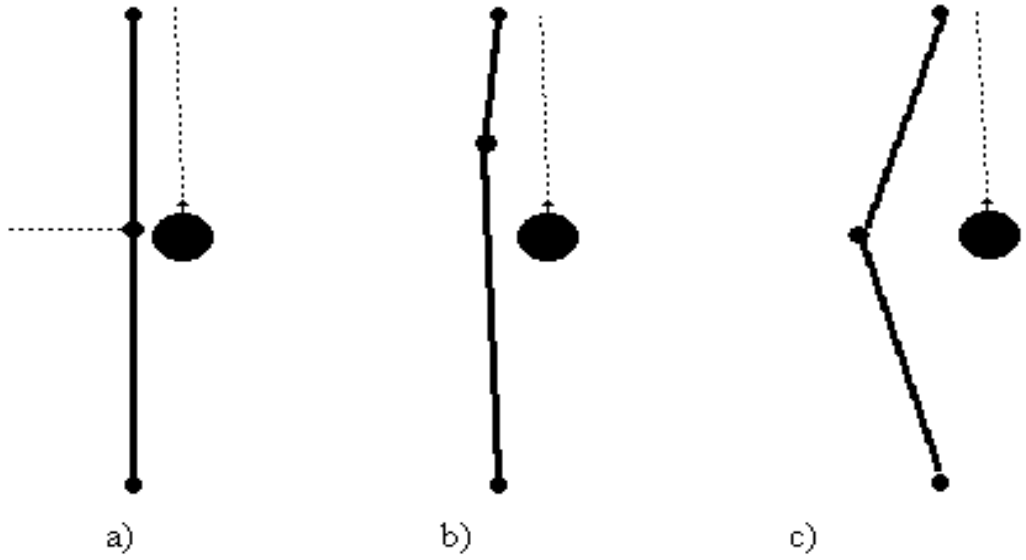

Figure 7.23 – Shows two different approaches to choosing a direction in which a vertex should be warped. Figure a) shows the tip of the gastroscope close to a vertex. The direction in which the tip of the gastroscope is pointed and the normal vector of the polygon are also shown. Figure b) shows how the vertex will be warped if it is warped in the direction of the tip of the gastroscope. Figure c) shows how the vertex will be warped using the normal vector of the polygon.

The second warping method was implemented and works as follows. Each vertex of each polygon in the list of polygons to be warped is inspected to find out if the vertex has already been warped. If not, it must be determined if the current vertex is a shared vertex and if it is, whether the other polygons that share the current vertex are also in the warp list. If not, these neighbour polygons must also be added to the warp list to ensure that no polygon is warped so that it is loose and not part of the VR model any more. The distance from each vertex to the tip of the gastroscope is then calculated. If this distance is less than the warp detection distance, the vertex must be warped away from the tip of the gastroscope in the direction opposite to its normal vector, so that it is away from the tip by the distance equal to the warp detection distance. At this stage in the procedure it is known *if* a vertex must be warped and in which *direction* it should be warped. The *distance* of how far the vertex must be warped, must still be calculated. The following paragraph will explain in detail how this distance is calculated.

Given the vertex **a** and the position of the tip of the gastroscope as point **b**, the vector **ab** can be translated to the origin to get a vector **AB**. Figure 7.24 shows the new vector with the normal vector of vertex **a** as **N**.

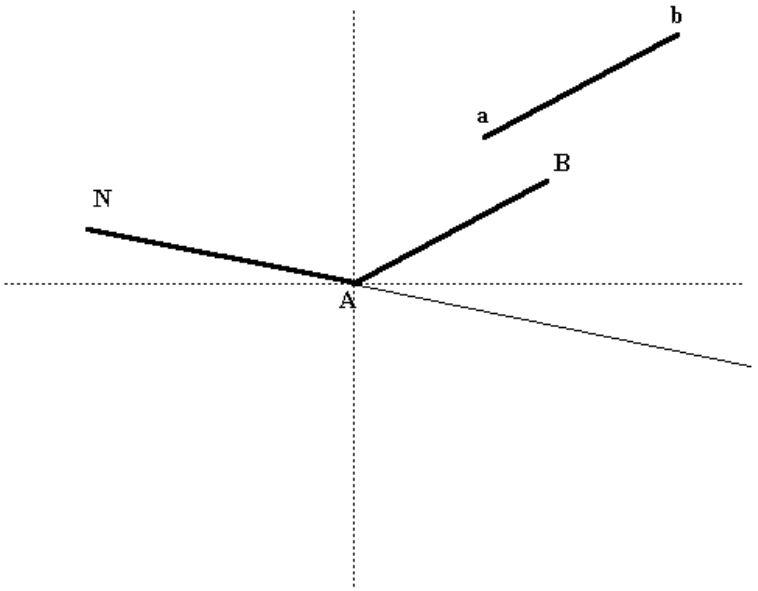

Figure 7.24 – Vertex **a** and the position of the tip of the gastroscope as point **b** forms a vector **ab**, which can be translated to the origin (0,0) as vector **AB**. **N** is the normal vector associated with vertex **a**.

To calculate the desired distance, it must be calculated how far **A** must be moved along the vector **N** so that the distance to **B** is equal to the warp detection distance. Simple trigonometric functions can be used for this.

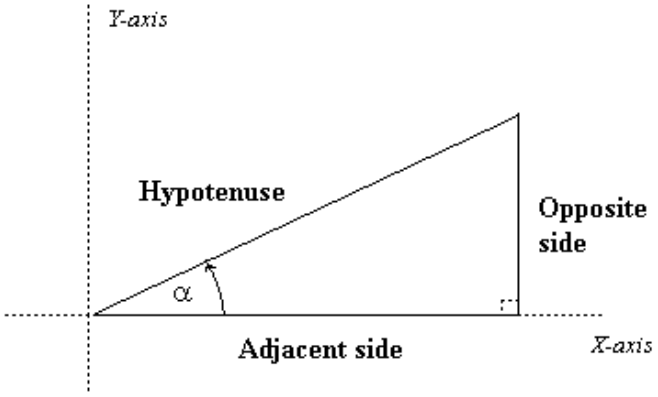

Figure 7.25 – Simple trigonometric functions can be formed using the above sketch.

$$Sin(\alpha) = \frac{Opposite}{Hypotenue} \tag{7-41}$$

$$Cos(\alpha) = \frac{Adjacent}{Hypotenuse} \tag{7-42}$$

$$Tan(\alpha) = \frac{Opposite}{Adjacent} \tag{7-43}$$

To use the above trigonometric functions a point **P** needs to be calculated, which is on the line **N** so that the line **PB** is orthogonal to the line **PN**, to form a triangle with a 90 degrees angle. Figure 7.26 illustrates where point **P** might be. To do this, **B** must be projected onto the vector **N**, which can be done using equation (7-44) [STR88].

$$\begin{bmatrix} P1 \\ P2 \end{bmatrix} = \frac{\begin{bmatrix} N1 & N2 \end{bmatrix} \begin{bmatrix} B1 \\ B2 \end{bmatrix}}{\begin{bmatrix} N1 & N2 \end{bmatrix} \begin{bmatrix} N1 \\ N2 \end{bmatrix}} \begin{bmatrix} N1 \\ N2 \end{bmatrix} \tag{7-44}$$

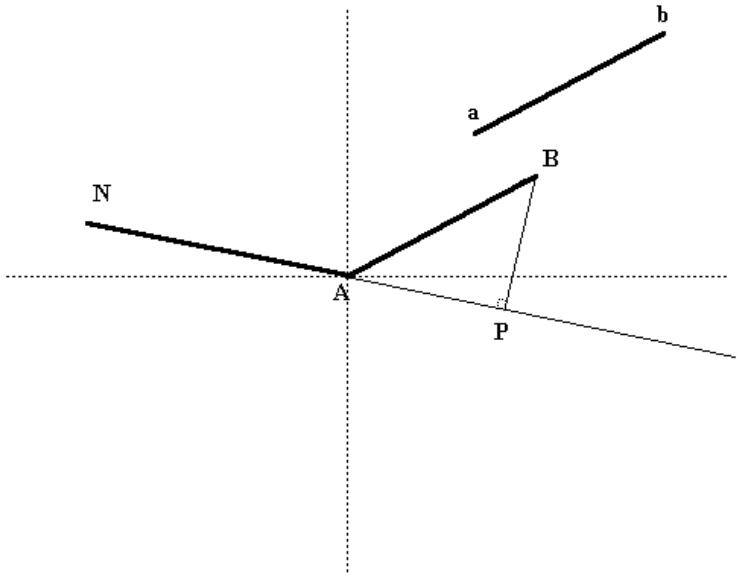

Figure 7.26 – Point **P** is a point on the line **N** so that the line **PB** is orthogonal to the line **PN**.

Now a point **Q** has to be calculated that is on line **PN**. This point is the new position of point **A** that should be the warp detection distance away from point **B**. See Figure 7.27.

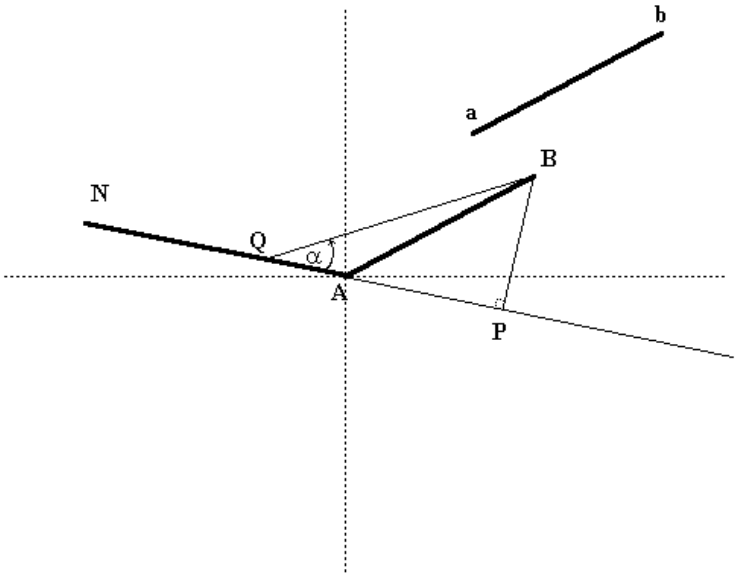

Figure 7.27 – Point **Q** is on the line **PN** which will be the new position of point **A**.

The length of **AQ** must now be calculated. The length of **QB** is known and the length of **PB** can be calculated using the distance formula. Since these two lengths are known, the angle α between **PQ** and **BQ** can be calculated by using equation (7-41). Knowing what the angle α is, equation (7-42) or (7-43) can be used to calculate the length of **PQ**. Because the angle between **PB** and **PQ** is 90 degrees, Pythagoras' formula can also be used, which is:

$$(DistAC)^2 = DistAB + DistBC \tag{7-45}$$

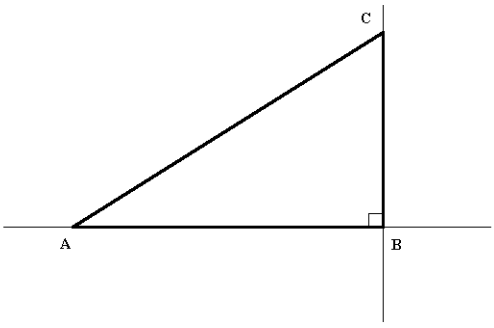

Figure 7.28 – A triangle that illustrates Pythagoras' distance formula.

The length of **PA** can be calculated using the distance formula. Therefore the length of **AQ** can be calculated by subtracting the length of **PA** from the length of **PQ**. This is the distance that the current vertex should be warped with, in the opposite direction of its own normal vector direction. Pseudo code for warping is as follows:

```
1. Get direction in which Camera looks at in vector: direction
2. Normalize direction
3. Get Camera position in cam_pos
4. for cnt = 0 to ListPos-1
  1. Get vertex indices of polygon stored at PolygonWarpList[cnt]
     in vertex_index    {get all indices of vertices in this polygon}
  2. for cnt2 = 0 to 3        {always 4 sided polygon}
   1. if VertexData[vertex_index[cnt2]-1] not warped  {then warp it}
    1. Get vertex, using vertex_index[cnt2] as index
    2. Get distance to camera position
     1. Distance = sqrt((cam_pos.x-vertex.x)² +
                        (cam_pos.y-vertex.y)² +
                        (cam_pos.z-vertex.z)² )
    3. if Distance <=  WarpRadius
     1. direction = SubtractVector (vertex,cam_pos)
     2. Normalize (direction)
     3. Dot = DotProduct (direction,
                          VertexData[vertex_index[cnt2]-1].Normal)
     4. if Dot < 0.0 then                 {they look at each other}
      1. A.x = -VertexData[v_index[cnt2]-1].Normal.x
      2. A.y = -VertexData[v_index[cnt2]-1].Normal.y
      3. A.z = -VertexData[v_index[cnt2]-1].Normal.z
      4. B.x = cam_pos.x-vertex.x
      5. B.y = cam_pos.y-vertex.y
      6. B.z = cam_pos.z-vertex.z
      7. TopEq = A.x*B.x + A.y*B.y + A.z*B.z
      8. BottomEq = A.x*A.x + A.y*A.y + A.z*A.z
      9. P.x = TopEq *A.x / BottomEq
      10. P.y = TopEq *A.y / BottomEq
      11. P.z = TopEq *A.z / BottomEq
      12. BP = sqrt ( (B.x-P.x)² +
                      (B.y-P.y)² +
                      (B.z-P.z)² )
      13. if BP <= Warp Detection Distance then       {use Pythagorus}
        1. QP = sqrt ((Warp Detection Distance)² - (BP)²)
        2. Q.x = P.x + A.x * QP
        3. Q.y = P.y + A.y * QP
        4. Q.z = P.z + A.z * QP
        5. AQ = sqrt ( (Q.x)² + (Q.y)² + (Q.z)² )
      14. else
        1. AQ = 0
     5. else
      1. AQ = 0
     6. if AQ <> 0 then                   {warp vertex using AQ}
      1. vertex.x = vertex.x + A.x * AQ
      2. vertex.y = vertex.y + A.y * AQ
      3. vertex.z = vertex.z + A.z * AQ
      4. VertexData[vertex_index[cnt2]-1].Warped = True
```

Pseudo code line 4 mentions a PolygonWarpList and ListPos. These were set up during the collision detection procedure. In the collision detection procedure, the distance from the tip of the gastroscope (cam_pos) to the centre of each polygon is inspected. In this procedure the distance to each vertex of each polygon in the PolygonWarpList is inspected. Line 4.2.3 checks to see if the vertex is within the warp radius and if it is, calculations have to be done to determine how far the vertex must be warped. Lines 4.2.3.1 to 4.2.3.3 calculate if the tip of the gastroscope's position is "beyond" the vertex by calculating a direction vector using the vertex as origin. Lines 4.2.3.4.1 to 4.2.3.4.11 correlate to Figure 7.26 and equation (7-44) that is used to determine the distance that the vertex should be warped.

The developed warping method gives satisfying results in terms of speed and in preventing the gastroscope from penetrating the VR model.

7.4.5 Transformed VR Model

The *transformed VR model* is the output of the *real-time transformation of the original VR model* module. The transformed VR model serves as input for the *render engine*. When training navigation skills, this model will serve as input to the *trainer daemon* so that the system can evaluate navigational tasks.

7.4.6 3-D Condition Database

The *3-D condition database* is a computer data set that is loaded into memory by the *initialisation* module. While editing a scenario, the available 3-D conditions are shown in a separate dialog box (the condition browser), so that the user can choose a condition. Plate 8.2 illustrates the condition browser. In Learn Mode, the *trainer daemon* uses the information from the database as input to show a condition's information. While in Test Mode, the *trainer daemon* uses information from this database as input to check if an answer by the user was correct or not. This database also serves as input for the *real-time transformation of the VR model* when a 3-D condition has to be deformed. For more detail about the 3-D condition database, see Chapter 6. The following sections will explain how the 3-D abnormal conditions are placed inside the VR model.

7.4.6.1 How the 3-D Conditions are Added to the VR Model

Adding 3-D conditions to the VR model changes the VR model since some vertices and polygons are removed from the VR model. Note that no changes are allowed to be made to the original VR model, therefore when adding 3-D conditions, they are *only* added to the transformed VR model. Two problems will be discussed in this section. The first is that the system should determine if a condition can be added to the VR model at a specific place, given the fact that 3-D conditions may not be placed over each other. The second problem is the actual adding of the 3-D condition once it is determined that the position where it must be added is an “available” position in the VR model.

Two situations can occur when adding a 3-D condition to the VR model. The first one is when the instructor adds the condition over only one polygon and the second is when the instructor adds the condition over an area of polygons. If conditions are added over only one polygon, then the system only needs to check whether that polygon has already been replaced by another condition. If it was not replaced, then that polygon is “available” and the condition can be added.

If the condition needs to be viewed larger and therefore be placed over an area of polygons, the instructor has to specify where or over which area of polygons in the VR model the condition should be placed. To specify where the condition must be placed, the instructor must use the mouse to select a desired area inside the VR model. This can be done by holding down the mouse button and dragging it over an area of polygons inside the VR model. 3-D conditions cannot be placed anywhere in the VR model, for example one 3-D condition cannot be placed on top of another. It can be replaced by first removing the first condition and then adding the second condition. Only the *visual* size of the 3-D condition will be set larger because the condition will be stretched over more than one polygon. Therefore the number of vertices and polygons from which the 3-D condition is modelled stays the same. When a 3-D condition has to be added over more than one polygon, the calculation of available polygons where it can be added gets complicated. The following paragraphs will explain in detail how the developed methods work to determine these available polygons.

The VR model's polygons are arranged like a grid. Figure 4.6 illustrates the polygons arranged as a grid. To help determine the available polygons, a 2-D array was therefore used since each position in the array can easily be correlated to a polygon in the VR model. Each time a 3-D condition is added, this array must be set up to verify valid placing positions for the 3-D condition. Each position in the array correlates with a polygon in the VR model. Most conditions are applied over only one or two polygons, but to provide for very large visual conditions a fixed sized 13x13 array was used. The size of this array must be an uneven number so that there will be a position exactly in the "middle" of the array which can be used for the polygon on which the user first clicked with the mouse. Note that for each condition that has to be placed, this 13x13 matrix will be set up. It will never change the condition size, only the visual size. For example, if the instructor clicked on a polygon and released the mouse button one polygon away from the first polygon, a 9x9 size 3-D condition will be placed over two polygons. Half of the 3-D condition will be applied to the one polygon and the other half to the other polygon.

The values in the array are of type integer. Each polygon in the VR model is tagged with a number by the render engine. The array values are set to the VR model's polygons' tags forming a 13x13 grid, as if the array was overlaid onto the selected area of the VR model with the middle of the array as the polygon on which the instructor first clicked. All the polygons in this 13x13 area of the VR model that were previously replaced by 3-D conditions will cause a value of –1 to be placed in the array. The –1's will indicate "unavailable" positions. For all 3-D conditions found in the array, all the array positions "behind" the found unavailable positions, facing from the centre of the matrix to the unavailable positions, should also be marked with –1's since these positions are also unavailable. Figure 7.29 illustrates how the matrix is divided into quadrants and explains the concept of one position being "behind" another one. See Figure 7.30 for an example of how the system determines where a condition can be added to the VR model.

Each time a condition is added, the array is initialised with zeros. The centre position of the array is defined as (6,6), indexing the array positions from 0 to 12. The value of the centre position is set to the tag of the polygon that was clicked on. The array can be divided into four quadrants, upper left, lower left, upper right and lower right. See Figure 7.29 for an illustration of these quadrants. The system looks for other 3-D objects that have

replaced polygons in the vicinity of the polygon in the middle of the array. With all four quadrants the same search must be done, but for each quadrant the search is in a different direction. For the upper left quadrant the search is left and up. In the lower left quadrant the search is left and down. In the upper right quadrant the search is right and up. In the lower right quadrant the search is right and down. For example, if a polygon is found in the upper left quadrant that was replaced by another 3-D condition, all the polygons forming a block to the left and upwards of this polygon are marked as unavailable. The instructor can drag the mouse to another available polygon. As soon as the mouse button is released, the 3-D condition will be placed over the area of available polygons.

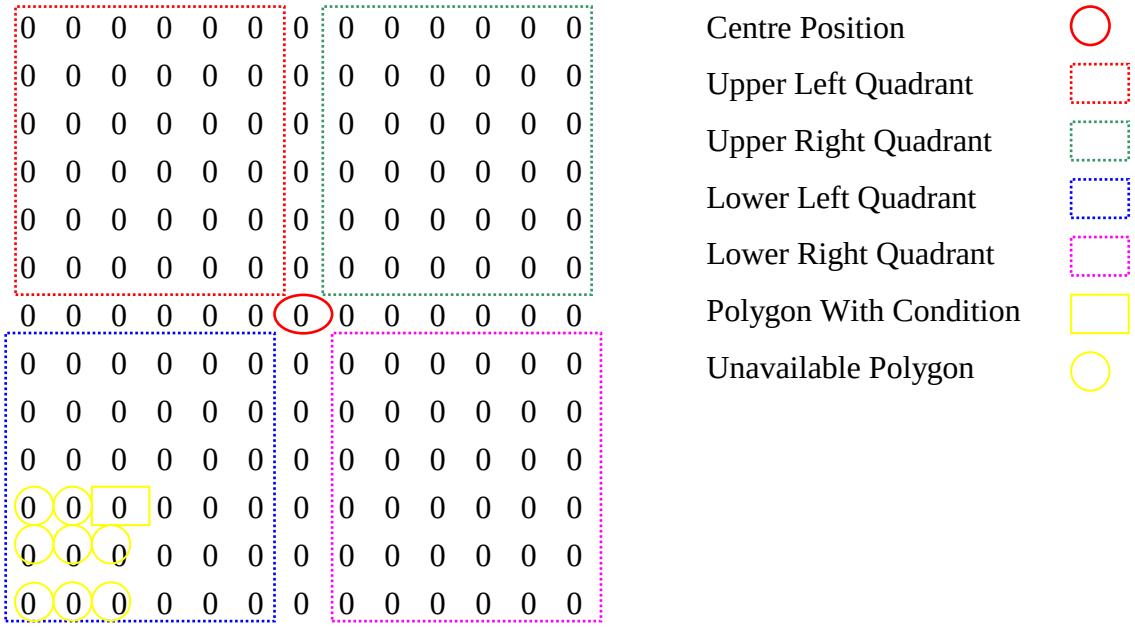

Figure 7.29 – A representation of a 13x13 array used to add 3-D conditions to the VR model. The four quadrants are shown, as well as the how the “behind” concept works. If the system finds a polygon that was replaced by a 3-D condition, it will be marked with a –1. The position marked with a yellow square will therefore be set to –1. All the positions “behind” this position, facing from the centre position to this position must also be marked as unavailable. Therefore all the positions marked with yellow circles will also be set to –1’s.

−1	−1	−1	−1	5	6	7	8	9	10	11	12	13
−1	−1	−1	−1	17	18	19	20	21	22	23	24	25
−1	−1	−1	−1	30	31	32	33	34	35	36	37	38
−1	−1	−1	−1	45	46	47	48	49	50	51	52	53
63	64	65	66	67	68	69	70	71	72	73	74	75
80	81	82	83	84	85	86	87	88	89	90	91	92
94	95	96	97	98	99	100	101	102	103	104	105	106
120	121	122	123	124	125	126	127	128	129	130	140	150
161	162	163	164	165	166	167	168	169	170	171	172	173
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1

Figure 7.30 – An example of how a 13x13 array can be used to calculate all available polygons in the VR model where a 3-D condition can be added. The blue circle indicates the centre position of the matrix with the polygon tag 100. The red circles show where other 3-D conditions have been placed.

If, for example, referring to Figure 7.30, the instructor clicked on the polygon with tag number 100, then the integer 100 is placed in the middle of the array. All the quadrants are then examined for unavailable polygons. In this case, the marked array positions (red circles) show where other 3-D conditions have been placed. Therefore, these positions and the positions “behind” them are not available. The instructor can therefore drag the mouse to any of the available polygons. The unavailable polygons will be darkened in the VR model so that the instructor can see where the available polygons are.

Pseudo code for calculating which polygons are available in the VR model follows:

```
1. Initialize PolygonGrid, a 13x13 array with zeros
2. PolygonGrid[6][6] = selected polygon's integer tag

{Now Search Upper Left Quadrant for unavailable grid positions}
3. for cnt = 0 to 5
 1. current_poly = PolygonGrid[6-cnt][6-cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6-cnt to 1                          {search to left}
   1. if PolygonGrid[cnt2-1][6-cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon left of current_poly
    3. if current_poly = -1                {set all to left to –1}
```

```
     1. current_poly = old_poly
     2. for gridx = 0 to cnt2-1
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1        {unavailable}
    4. else
     1. PolygonGrid[cnt2-1][6-cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6-cnt][6-cnt]
  3. for cnt2 = 6-cnt to 1                            {search up}
   1. if PolygonGrid[6-cnt][cnt2-1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on top of current_poly
    3. if current_poly == -1              {set all upwards to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to 12
      1. for gridy = 0 to cnt2-1
       1. PolygonGrid[gridx][gridy] = -1        {unavailable}
    4. else
     1. PolygonGrid[6-cnt][cnt2-1] = current_poly     {available}
  4. if PolygonGrid[6-cnt-1][6-cnt-1] <> -1
   1. PolygonGrid[6-cnt-1][6-cnt-1] = polygon in the upper left
         corner of PolygonGrid[6-cnt][6-cnt]         {available}
   2. if PolygonGrid[6-cnt-1][6-cnt-1] == -1
    1. for gridx = 0 to 6-cnt-1
     1. for gridy = 0 to 6-cnt-1
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    2. cnt = 6  (finished with quadrant)

{Now Search Lower Left Quadrant for unavailable grid positions}
4. for cnt = 0 to 5
 1. current_poly = PolygonGrid[6-cnt][6+cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6-cnt to 1               {search to the left}
   1. if PolygonGrid[cnt2-1][6+cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon left of current_poly
    3. if current_poly = -1               {set all to left to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to cnt2-1
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[cnt2-1][6+cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6-cnt][6+cnt]
  3. for cnt2 = 6+cnt to 11                           {search Down}
   1. if PolygonGrid[6-cnt][cnt2+1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on underneath current_poly
    3. if current_poly =-1                {set all downward to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to 12
      1.for gridy = cnt2+1 to 12
       1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[6-cnt][cnt2+1] = current_poly     {available}
  4. if (PolygonGrid[6-cnt-1][6+cnt+1] != -1)
   1. PolygonGrid[6-cnt-1][6+cnt+1] = polygon in the lower left
         corner of PolygonGrid[6-cnt][6+cnt]
   2. if PolygonGrid[6-cnt-1][6+cnt+1] == -1
    1. for gridx = 0 to 6-cnt-1
```

```
     1. for gridy = 6+cnt+1 to 12
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    2. cnt = 6 (finished with quadrant)
 3. PolygonGrid[0][12] = -1

{Search Lower Right Quadrant for unavailable grid positions}
5.for cnt = 0 to 6
 1. current_poly = PolygonGrid[6+cnt][6+cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6+cnt to 11                      {search to the right}
   1. if PolygonGrid[cnt2+1][6+cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon to the right of current_poly
    3. if current_poly =-1               {set all to right to –1}
     1. current_poly = old_poly
     2. for gridx = cnt2+1 to 12
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1             {unavailable}
    4. else
     1. PolygonGrid[cnt2+1][6+cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6+cnt][6+cnt]
  3. for cnt2 = 6+cnt to 11                     {search down}
   1. if PolygonGrid[6+cnt][cnt2+1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on underneath current_poly
    3. if current_poly = -1              {set all downwards to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to 12
      1. for gridy = cnt2+1 to 12
       1. PolygonGrid[gridx][gridy] = -1             {unavailable}
    4. else
     1. PolygonGrid[6+cnt][cnt2+1] = current_poly     {available}
  4. if PolygonGrid[6+cnt+1][5+cnt+1] <> -1
   1. PolygonGrid[6+cnt+1][6+cnt+1] = polygon in the lower right
                          corner of PolygonGrid[6+cnt][6+cnt]
   2. if PolygonGrid[6+cnt+1][6+cnt+1] = -1
    1. for gridx = 6+cnt+1 to 12
     1. for gridy = 6+cnt+1 to 12
      1. PolygonGrid[gridx][gridy] = -1             {unavailable}
    2. cnt = 6                                  {finished with quadrant}

{Now Search Upper Right Quadrant for unavailable grid positions}
6. for cnt = 0 to 5
 1. current_poly = PolygonGrid[6+cnt][6-cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6+cnt to 11              {search to the right}
   1. if PolygonGrid[cnt2+1][6-cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon to the right of current_poly
    3. if current_poly =-1                      {set all to right to –1}
     1. current_poly = old_poly
     2. for gridx = cnt2+1 to 11
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1             {unavailable}
    4. else
     1. PolygonGrid[cnt2+1][6-cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6+cnt][6-cnt]
  3. for cnt2=6-cnt to 1                            {search up}
   1. if PolygonGrid[6+cnt][cnt2-1] <> -1
    1. old_poly = current_poly
```

```
   2. current_poly = polygon on top of current_poly
   3. if current_poly =-1                      {set all upward to –1}
    1. current_poly = old_poly
    2. for gridx = 0 to 12
     1. for gridy = 0 to cnt2-1
      1. PolygonGrid[gridx][gridy] = -1            {unavailable}
   4. else
    1. PolygonGrid[6+cnt][cnt2-1] = current_poly     {available}
 4. if PolygonGrid[6+cnt+1][6-cnt-1] <> -1
  1. PolygonGrid[6+cnt+1][6-cnt-1] = polygon in the upper right
                                corner of PolygonGrid[6+cnt][6-cnt]
  3. if PolygonGrid[6+cnt+1][6-cnt-1] = -1
   1. for gridx = 6+cnt+1 to 12
    1. for gridy = 0 to 6-cnt-1
     1. PolygonGrid[gridx][gridy] = -1              {unavailable}
   2. cnt = 6                                {finished with quadrant}
3. PolygonGrid[12][0] = -1
```

Pseudo code line 3 shows the checking of the upper left quadrant, line 4 the lower left quadrant, line 5 the lower right quadrant and line 6 the upper right quadrant.

Each 3-D condition replaces one or more polygons in the VR model. These polygons are deleted from the VR model but the information is stored in original VR model in case the user wants to remove the condition. The VR model has a hole where the polygon or polygons have been removed. The selected 3-D condition must be rotated, translated and scaled to fit exactly into this hole. Plate 7.10 illustrates a hole in the VR model where a 3-D condition must be fitted. Plate 7.11 illustrates how the 3-D condition was fitted into the hole. Considering how many calculations would be needed to rotate, translate and scale a 3-D condition to fit *exactly* into a removed polygon's hole, this method would be too slow when it comes to warping 3-D conditions in real-time. Refer to section 7.4.6.2 for warping of 3-D conditions. The following paragraphs will explain how this problem was overcome in this system.

A 3-D condition is generated by generating a flat 3-D grid in the XZ-plane. The only thing that makes a flat 3-D grid different from a hill-like 3-D grid is the vertices' heights in the Y direction of the hill-like 3-D grid. Therefore, the heights of a 3-D condition's vertices are important to give it a 3-D look. Instead of calculating the correct rotation, translation and scaling for a 3-D condition, a new method was developed which involves the following:

A new 3-D grid can be generated that fits into the hole left by the removed polygons. The new 3-D grid's vertices can be generated using the vertices of the hole in the VR model and applying the heights of the 3-D condition to the newly generated 3-D grid. This will then fill the hole in the VR model exactly. This method also has the advantage that when the VR model's vertices are warped, the 3-D condition will always fit in the hole because the vertices of the borders of the hole in the VR model are used to regenerate the grid each time the VR model is deformed.

In the case where only one polygon is replaced, the heights of the 3-D condition's vertices are applied to the new 3-D grid's vertices in the same direction as the deleted polygon's normal vector. In the case where more than one polygon is deleted, it is most probable that the deleted polygons form a curved surface. In the latter case, the 3-D condition grid has to be fitted evenly into the hole. In other words, if the hole was made by deleting two polygons, then half of the 3-D condition must be fitted in the one polygon's hole and half of the 3-D condition must be fitted in the other polygon's hole. This makes things more complicated since separate grids with different sizes have to be generated for each polygon that was deleted. The heights of the 3-D condition's vertices are applied to the grid's vertices in the same direction as the deleted polygons' *average* normal vector. The following paragraphs will explain in detail the generating of a new 3-D grid so that in the end it looks exactly like the 3-D condition that was selected, from the condition browser, to be added to the VR model.

Figure 7.31 shows part of the VR model. The more dense part of the picture is the 3-D grid generated between the border vertices of the polygon that was deleted. The grid is generated by using the grid size (rows, columns) of the 3-D condition grid that has to be added to the VR model. The image applied to the 3-D condition is also applied to the new 3-D grid.

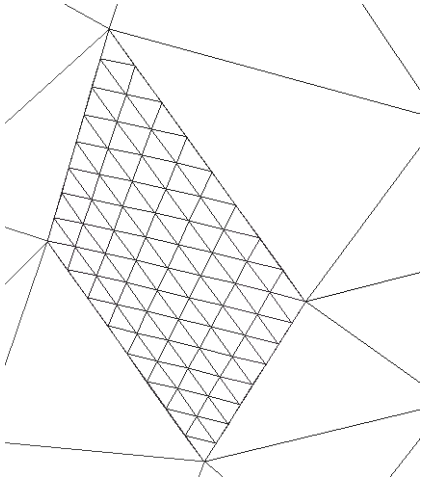

Figure 7.31 – Grid generated to fit into the polygon space that was removed.

After the grid has been generated, the heights of the 3-D condition grid must be applied to the new grid. Using the normal vector of the removed polygon, or the average normal vector if more than one polygon were removed, for the direction in which the heights have to be applied, the y-coordinate of each vertex in the new 3-D grid can be set according to the y-coordinate of the 3-D condition. The polygons in the VR model will not all be the same size or shape which will leave holes with different sizes when a condition has to be added. The heights of the 3-D condition therefore cannot just be applied to the new 3-D grid. A scaling factor needs to be calculated to ensure that the new 3-D grid looks like the 3-D condition in proportion, even if the hole where the grid was generated in is much smaller or bigger than the 3-D condition.

Calculating the surface of the base of the 3-D condition and the sum of the surfaces of the deleted polygons, a very accurate scaling factor can be calculated. To calculate the surface of a polygon, the polygon needs to be triangulated and the sum of the surfaces of the triangles will be the total surface of the polygon. Referring to the design specifications of the VR model in this system, every polygon must have four vertices. To calculate the surface of such a polygon in 3-D space, the polygon needs to be divided into two triangles. Each triangle's surface can be calculated using the formula Surface = ½ * Base * Height. A scaling factor can then be calculated using Scaling Factor = Total Surface of New 3-D Gid / Total Surface of 3-D Condition.

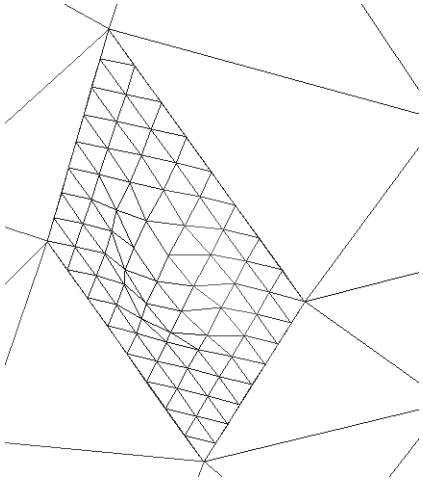

Figure 7.32 – Applying the heights of each grid point in the correct direction.

At this point the new 3-D grid should look like the 3-D condition, but the lighting of the VR model creates two other problems. Figure 7.32 shows the wireframe model of a 3-D condition added to the VR model. The first problem is quite obvious. For smooth shading the average normal vector of all the vertices sharing a 3-D coordinate has to be calculated. But since the 3-D condition only shares its corner vertices with the VR model polygons' vertices, only the corners will be smoothed into the VR model. The polygons in the middle of the condition will also be smooth, but not those between the corners on the borders, touching the VR model. To solve this problem a strip of triangles can be added around the newly generated 3-D grid as a border between the VR model and the 3-D condition. These triangles will be referred to as the border triangles. The grid has to be generated smaller to make provision for the border triangles. For each side, 25% of the border was used for the border triangles.

Figure 7.33 illustrates the smaller grid without the strip of border triangles between the polygons of the VR model and the 3-D condition.

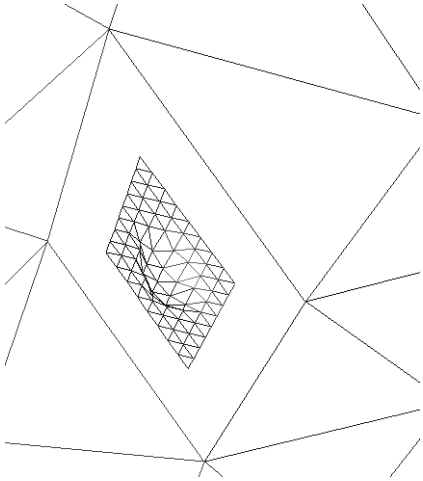

Figure 7.33 – The 3-D condition *without* border triangles, but generated smaller to make provision for border triangles.

If the border is smaller, the change from the higher density vertices (the new 3-D grid) to the lower density vertices (the VR model) is too sudden, and the smooth effect is lost again. The use of border triangles of course affects the scaling of the heights of the 3-D grid since the surface where the 3-D condition is now fitted, is smaller. Figure 7.34 shows how the triangle border is generated. Because of the triangle border, texture mapping also has to be changed from that of the 3-D condition since the border triangles should also display part of the texture. If this is not done, the 3-D condition will "use" all of the texture, then there will be a border of triangular polygons with no texture next to the VR model's polygons with textures.

The second problem is that the density of vertices in the 3-D condition is much higher than in the VR model causing the 3-D condition to be illuminated much more since there is more information available for calculating average normal vectors. What happens is that where a 3-D condition is displayed, the 3-D condition shines much more, or reflects much more light than the polygons surrounding it and is easily seen. To make the condition blend more into the background of the VR model, the intensity and saturation of the polygons in the 3-D condition had to be set a little less. This was set by trial and error until the best results were obtained. The easiest way of obtaining satisfactory results was to set the intensity and saturation very low and then set it higher and higher until the overall light reflection in the VR model was good enough and 3-D conditions blended in well enough.

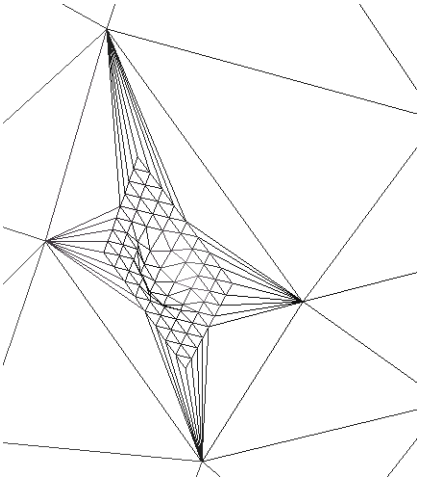

Figure 7.34 – The 3-D condition fitted into a removed polygon's space. The use of border triangles can clearly be seen. The border triangles are four bigger ones as well as all the smaller ones surrounding the 3-D condition.

Pseudo code for adding a 3-D condition to the VR model follows:

```
1. Calculate original 3-D condition surface for scaling by:
 1. Get LL lower left vertex of condition
 2. Get UL upper left vertex of condition
 3. Get UR upper right vertex of condition
 4. Get LR lower right vertex of condition
 5. original_surface = CalcTriangleSurface(UL,LL,LR) +
                       CalcTriangleSurface(UL,UR,LR)
2. Calculate surface of polygon to be replaced by condition by:
 1. Get LL lower left vertex of polygon
 2. Get UL upper left vertex of polygon
 3. Get UR upper right vertex of polygon
 4. Get LR lower right vertex of polygon
 5. poly_surface = CalcTriangleSurface(UL,LL,LR) +
                   CalcTriangleSurface(UL,UR,LR)
3. Calculate scaling factor for fitted condition's grid heights by:
 1. scaling_factor = poly_surface/original_surface
4. Get normal vectors of polygon vertices by:
 1. Get LL_normal lower left vertex of polygon
 2. Get UL_normal upper left vertex of polygon
 3. Get UR_normal upper right vertex of polygon
 4. Get LR_normal lower right vertex of polygon
5. Calculate direction vectors for each side of polygon by:
 1. Left_Dir = SubtractVector (UL,LL)
 2. Right_Dir = SubtractVector (UR,LR)
 3. Upper_Dir = SubtractVector (UR,UL)
 4. Lower_Dir = SubtractVector (LR,LL)
 5. Normalize (Left_Dir)
 6. Normalize (Right_Dir)
 7. Normalize (Lower_Dir)
 8. Normalize (Upper_Dir)
6. Calculate distance for upper and lower sides by:
 1. Upper_Dist = sqrt ( (UR.x-UL.x)² + (UR.y-UL.y)² + (UR.z-UL.z)² )
 2. Lower_Dist = sqrt ( (LR.x-LL.x)² + (LR.y-LL.y)² + (LR.z-LL.z)² )
```

7. Calculate Border points LLB and ULB by:(Lower Left and Upper Left)
 1.Get Lower point
  1. LPoint.x = LL.x + Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.x
  2. LPoint.y = LL.y + Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.y
  3. LPoint.z = LL.z + Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.z
 2. Get Upper point
  1. UPoint.x = UL.x + Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.x
  2. UPoint.y = UL.y + Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.y
  3. UPoint.z = UL.z + Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.z
 3. Calculate direction vector between the upper and lower points
  1. Left_Dir = SubtractVector (UPoint,LPoint)
  2. Normalize (Left_Dir)
 4. Calculate distance for left side border points
  1. Left_Dist = sqrt ((UPoint.x-LPoint.x)² +
                        (UPoint.y-LPoint.y)² +
                        (UPoint.z-LPoint.z)²)
 5. LLB.x = LPoint.x + Left_Dist * COND3D_BORDER / 200 * Left_Dir.x
 6. LLB.y = LPoint.y + Left_Dist * COND3D_BORDER / 200 * Left_Dir.y
 7. LLB.z = LPoint.z + Left_Dist * COND3D_BORDER / 200 * Left_Dir.z
 8. ULB.x = UPoint.x - Left_Dist * COND3D_BORDER / 200 * Left_Dir.x
 9. ULB.y = UPoint.y - Left_Dist * COND3D_BORDER / 200 * Left_Dir.y
 10. ULB.z = UPoint.z - Left_Dist * COND3D_BORDER / 200 * Left_Dir.z
8. Calculate Border points, LRB,URB by: (Lower Right, Upper Right)
 1. Get Lower point
  1. LPoint.x = LR.x - Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.x
  2. LPoint.y = LR.y - Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.y
  3. LPoint.z = LR.z - Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.z
 2. Get Upper point
  1. UPoint.x = UR.x - Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.x
  2. UPoint.y = UR.y - Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.y
  3. UPoint.z = UR.z - Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.z
 3. Calculate direction vector between the upper and lower points by:
  1. Right_Dir = SubtractVector (UPoint,LPoint)
  2. Normalize (Right_Dir)
 4. Calculate distance between upper and lower points by:
  1. Right_Dist = sqrt ((UPoint.x-LPoint.x)² +
                         (UPoint.y-LPoint.y)² +
                         (UPoint.z-LPoint.z)²)
 5. LRB.x = LPoint.x + Right_Dist * COND3D_BORDER / 200 * Right_Dir.x
 6. LRB.y = LPoint.y + Right_Dist * COND3D_BORDER / 100 * Right_Dir.y
 7. LRB.z = LPoint.z + Right_Dist * COND3D_BORDER / 100 * Right_Dir.z
 8. URB.x = UPoint.x - Right_Dist * COND3D_BORDER / 100 * Right_Dir.x
 9. URB.y = UPoint.y - Right_Dist * COND3D_BORDER / 100 * Right_Dir.y
 10. URB.z =UPoint.z - Right_Dist * COND3D_BORDER / 100 * Right_Dir.z
9. Upper_Step = sqrt ((URB.x-ULB.x)² +
                       (URB.y-ULB.y)² +
                       (URB.z-ULB.z)²) / (URx-LLx)
10. Lower_Step = sqrt ((LRB.x-LLB.x)² +
                        (LRB.y-LLB.y)² +
                        (LRB.z-LLB.z)²) / (URx-LLx)
11. Transfor condition grid points into polygon space by:
 1. for cnt = 0 to URx-LLx   (horizontal)
  1. Get Lower point by:
   1. LPoint.x = LLB.x + (cnt * Lower_Step * Lower_Dir.x)
   2. LPoint.y = LLB.y + (cnt * Lower_Step * Lower_Dir.y)
   3. LPoint.z = LLB.z + (cnt * Lower_Step * Lower_Dir.z)
  2. Get Upper point by:
   1. UPoint.x = ULB.x + (cnt * Upper_Step * Upper_Dir.x)

```
    2. UPoint.y = ULB.y + (cnt * Upper_Step * Upper_Dir.y)
    3. UPoint.z = ULB.z + (cnt * Upper_Step * Upper_Dir.z)
   3. Calculate direction vector between the upper and lower by:
    1. Left_Dir = SubtractVector (UPoint,LPoint)
    2. Normalize (Left_Dir)
   4. Calculate distance and step distance for each side by:
    1. Left_Step = sqrt ( (UPoint.x-LPoint.x)² +
                          (UPoint.y-LPoint.y)² +
                          (UPoint.z-LPoint.z)² ) / (URy-LLy)
   5. for cnt2 = 0 to URy-Lly                              (vertical)
    1. Calculate flat grid by:
     1. grid[cnt][cnt2].x = LPoint.x + (cnt2 * Left_Step * Left_Dir.x)
     2. grid[cnt][cnt2].y = LPoint.y + (cnt2 * Left_Step * Left_Dir.y)
     3. grid[cnt][cnt2].z = LPoint.z + (cnt2 * Left_Step * Left_Dir.z)
    2. Add the height of each grid point by:
     1. vert_cnt = (cnt * grid_size) + cnt2
     2. Get original grid height of vertex with index vert_cnt+1
     3. Height = original_height * scaling_factor
     4. grid[cnt][cnt2].x = grid[cnt][cnt2].x + (Height *avg_normal.x)
     5. grid[cnt][cnt2].y = grid[cnt][cnt2].y + (Height *avg_normal.y)
     6. grid[cnt][cnt2].z = grid[cnt][cnt2].z + (Height *avg_normal.z)
    3. Add texture map info by:
     1. U[cnt][cnt2] = ( cnt / (grid_size-1)) *
                       ( (100-COND3D_BORDER)/100) +
                       (COND3D_BORDER / 100)
     2. V[cnt][cnt2] = ( cnt2 / (grid_size-1)) *
                       ( (100-COND3D_BORDER)/100) +
12. Add UL,LL, UR and LR vertices of polygon to condition
13. Set LL texture map info to 0,0
14. Set UL texture map info to 0,1
15. Set UR texture map info to 1,1
16. Set LR texture map info to 1,0
17. Add left border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor-2
  1. triangle_vertices[0] = LL_Index
  2. triangle_vertices[1] = cnt+1
  3. triangle_vertices[2] = cnt+2
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. triangle_vertices[0] = LL_Index
 4. triangle_vertices[1] = factor
 5. triangle_vertices[2] = UL_Index
 6. Add new polygon, using triangle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = UL_Index
  2. triangle_vertices[1] = cnt+1
  3. triangle_vertices[2] = cnt+2
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
18. Add top border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor-2
  1. triangle_vertices[0] = UL_Index
  2. triangle_vertices[1] = (cnt*grid_size)+ grid_size
  3. triangle_vertices[2] = (cnt+1)*(grid_size) + grid_size
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
```

```
 3. vertices[0] =UL_Index
 4. vertices[1] =((LLx+factor-1)*(grid_size))+ URy+1 //-1+1
 5. vertices[2] =UR_Index
 6. Add new polygon, using triangle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = UR_Index
  2. triangle_vertices[1] =(cnt*grid_size) + grid_size
  3. triagnle_vertices[2] = (cnt+1)*grid_size + grid_size
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
19. Add right border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor-2
  1. triangle_vertices[0] = UR_Index
  2. triangle_vertices[1] =((grid_size-1)*grid_size)+ grid_size-cnt
  3. triangle_vertices[2] =((grid_size-1)*grid_size)+ grid_size-1-cnt
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. triangle_vertices[0] = UR_Index
 4. triangle_vertices[1]=((grid_size-1)*grid_size)+grid_size-factor-1
 5. triangle_vertices[2] = LR_Index
 6. Add new polygon, using triangle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = LR_Index
  2. triangle_vertices[1] =((grid_size-1)*grid_size)+ grid_size-cnt
  3. triangle_vertices[2] =((grid_size-1)*grid_size)+ grid_size-1-cnt
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
20. Add bottom border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor - 2
  1. triangle_vertices[0] = LR_Index
  2. triangle_vertices[1] =((grid_size-1-cnt)*grid_size) + 1
  3. triangle_vertices[2] =((grid_size-cnt-2)*grid_size) + 1
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. triangle_vertices[0] = LR_Index
 4. triangle_vertices[1] =((grid_size-factor-2)*grid_size) + 1
 5. triangle_vertices[2] = LL_Index
 6. Add new polygon, using tringle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = LL_Index
  2. triangle_vertices[1] =((grid_size-cnt)*grid_size) + 1
  3. triangle_vertices[2] =((grid_size-cnt-2)*grid_size) + 1
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
```

Pseudo code lines 1 to 3 calculate a scaling factor that will be used to apply to the heights of the vertices in the 3-D condition. Line 7.1.1 mentions a constant COND3D_BORDER. This is the 25% border mentioned earlier in this section. Line 11 generates the 3-D vertices of the condition in the 3-D space of the hole left by deleting the polygon. Lines 17 to 20 generate the border triangles. Lines 1.5 and 2.5 mention the calculation of the surface of a 3-D triangle. Pseudo code for calculating the surface of a 3-D triangle follows:

```
1. Translate triangle to origin by:
 1. B = SubtractVector (b,c)
 2. A = SubtractVector (a,c)
2. Calculate a point P which lies on the base vector A
    so that vector BP is orthogonal on A
 1. P.x = (A.x*B.x + A.y*B.y + A.z*B.z)*A.x/
          (A.x*A.x + A.y*A.y + A.z*A.z)
 2. P.y = (A.x*B.x + A.y*B.y + A.z*B.z)*A.y/
          (A.x*A.x + A.y*A.y + A.z*A.z)
 3. P.z = (A.x*B.x + A.y*B.y + A.z*B.z)*A.z/
          (A.x*A.x + A.y*A.y + A.z*A.z)
3. Calculate triangle Height by:
 1. Height = sqrt ( (P.x-B.x)² +
                    (P.y-B.y)² +
                    (P.z-B.z)² )
4. Calculate triangle Base by:
 1. Base = sqrt ( (A.x)² + (A.y)² + (A.z)² )
5. Surface = 0.5*Base*Height
```

7.4.6.2 Warping 3-D Conditions

When the tip of the gastroscope gets too close to the surface of the VR model, the VR model's polygons have to be warped in a certain direction so that the tip of the gastroscope cannot penetrate the VR model. When some of these polygons that have to be warped have been replaced by 3-D conditions, the 3-D conditions also have to be warped in the same way that the polygons would have been warped. Warping 3-D conditions is almost the same procedure as adding 3-D conditions. What basically happens is that each time a polygon surrounding a 3-D condition is warped, the 3-D condition is "refitted" or "re-added" to the hole that was left by the removed polygon(s). This has to be done because warping the surrounding polygons will change the form and size of the hole where the 3-D condition is supposed to be fitted. The only difference between adding a new 3-D condition and warping an existing one is that no user interaction is needed with warping. Consequently no array has to be set up to test for valid positions in the VR model.

7.4.7 Region Database

This database is one of the computer data sets that are loaded into memory by the *initialisation* module when the system starts. It contains information about the different regions inside the VR model and by which polygons each region is defined. This database serves as input for the t*rainer daemon* while the user is trying to complete a navigational task. It also serves as input for the r*eal-time transformation of the VR model* when in Learn Mode to show the current region.

7.4.8 Trainer Daemon

Since this is a training simulator, the *trainer daemon* is a very important module of the system. In this system, the *trainer daemon* monitors certain events that occur while a trainee is working with the simulator, so that the trainee can be taught how gastroscopic procedures should be done. The daemon monitors the user interaction continuously to see whether the user has done something to be evaluated. This module gets its input from several other modules in the system. Two main events, one for identification skills training and one for navigation skills training, can occur. These events are monitored by the *trainer daemon*. These events only occur when the system is not in Edit Mode because the system does not need to monitor user activity while the instructor is setting up a scenario. This means that the daemon is not active in Edit Mode. The following sections will explain how the trainer daemon works and how it was implemented.

The daemon is a timer function that is active as long as the system is not in edit mode. It waits for certain events to occur so that it can evaluate the event. Basically an event can occur for the identification skills training or for the navigation skills training.

The identification skills training event occurs when the trainee clicks with the mouse on a 3-D condition in the main window. When the system is in Learn Mode, this module gets information about a specific 3-D condition from the 3-D condition database and shows the information on the screen. If the system is in Test Mode, this module generates a multiple-choice question from the 3-D condition database and waits for input from the user interface to evaluate the answer. When an answer is correct, the trainee scores a point. When an answer is incorrect, the incorrect answer will be saved so that it can be shown in the evaluation report. Each 3-D condition can only be evaluated once, otherwise a trainee may easily keep guessing what the correct answer is.

The navigation skills training event occurs with each rendered frame. Each time a frame is rendered, the position of the tip of the gastroscope is examined and compared with the current task in the task list. Currently three different tasks can be composed. Examining of a selected anatomical region, biopsy sampling on marked regions and cauterisation (“To burn or sear so as to stop bleeding and prevent infection” [ENC97]) of marked regions.

For the examining of an anatomical region, each polygon in the selected region must have been examined, meaning that the polygon must at some stage have been close enough to the tip of the scope *and* in the field of view of the scope. The region database is used to confirm which polygons are part of a certain anatomical region. In Learn Mode, the selected region is shown in the main window and orientation window by rendering these polygons darker (less light reflection) than the other polygons. In Test Mode, the selected region is not shown and the trainee should know where the region is. This evaluation was implemented using a TRUE / FALSE switch for whether each polygon was examined or not. Polygons facing the tip of the gastroscope and close to the tip of the gastroscope can be identified in the collision detection procedure. If all the polygons in the region were examined, the region was examined.

Cauterisation of a certain marked region is basically the same as the examining of anatomical regions. Care must be taken not to confuse the terms “marked region” and “anatomical region”. An anatomical region is the region that is defined in the region database and consists of several polygons. A marked region is a polygon or 3-D condition on which the instructor has to click to mark it as the specific place where a biopsy or cauterisation must be done. One of the differences between examining an anatomical region and the cauterisation of a marked region is that only one polygon or 3-D condition can be marked as a marked region for cauterisation, so only one polygon or 3-D condition has to be checked. The other difference is that the tip of the gastroscope must almost touch the marked region, unlike examining anatomical regions where the tip can be a certain distance from the polygon. This touching of the scope is determined during the collision detection procedure. Implementation of this task was also done using the collision detection procedure. If one of the polygons that were placed in the polygon warp list is the marked polygon, then the trainer daemon identifies the task as being complete.

Doing a biopsy on a marked region is exactly the same as cauterising a marked region, except that the biopsy forceps must be open when being too close, almost touching the marked region and closed while being the tool is taken away from the marked region to simulate the grabbing of tissue.

Evaluation is not as easy as with the identification skills where an answer can only be correct or incorrect. Firstly, the task to evaluate must have been completed correctly, for example the biopsy should have been taken on the correct marked region. Secondly the daemon times how long it takes a trainee to complete each task. The time is only used to show in the evaluation report so that an instructor can see if a trainee progresses or not. It will not be used to allocate higher or lower ratings for several reasons: The system cannot know how long a certain task *should* take, since time is not really a critical factor with endoscopy. A trainee should rather be relaxed and do the procedure thoroughly than rush it. If for example a higher rating is given to a procedure that was done quicker, then a trainee will automatically want to finish the task as quickly as possible, which could lead to neglecting critical observations. Therefore the time taken should only be seen as a progress indicator by the instructor.

A trainee evaluation is defined as a number of tests. A test can comprise of identification and/or navigation skills. The result of identification skills tests is a percentage mark using the number of correct and incorrect answers generated by the training daemon. For each incorrect answer it will also show why it was incorrect. The result of navigation skills tests is a percentage based on how many tasks have been completed in each task list and how long it took the trainee to complete it. Plate 8.11 shows an example of an evaluation report.

7.4.8.1 Navigational Training Task Lists

A task list is defined as a list of navigational tasks that a trainee must perform. A task has two parts, the action to be performed and a region on which the action must be performed. For example, “examine pylorus”, “examine” is the action part and “pylorus” is the region on which the action must be performed. Tasks can be constructed in the Edit Mode. Two dropdown lists can be used. The first one contains all the actions that can be chosen from. The second one contains, relative to the action chosen, all the valid regions on which the action can be performed. These task actions are predefined, but in later versions it could be user defined. Once an action and region have been selected using the dropdown lists, the text of the action and the region is combined and inserted into another dropdown list, which will serve as the task list. This list is maintained by the operating system since it is

part of a dropdown list and will be used as input for the trainer daemon for evaluation. Using these tasks, a scenario can be set up and saved. The main purpose of the task lists is to set up structured tasks that can be evaluated by the trainer daemon.

7.4.9 Render Engine

The *render engine* takes as input all the 3-D objects in a 3-D scene, and calculates the 2-D image from the 3-D camera's viewpoint, making use of the positions of the camera and light source(s) in the scene. The output of this module is then relayed to the *computer graphics user interface*

7.5 Conclusion

The purpose of this chapter was to explain how the virtual reality system works and how it was implemented. Figure 3.2 shows that the virtual reality system is not the only part of this system, but that many other components and software applications are needed with the virtual reality system to form a complete and successful system. Some problems discussed in this chapter are force feedback, collision detection, deforming the shape of the VR model, adding 3-D conditions in the VR model and using the trainer daemon for evaluation purposes.

Using natural force feedback proved to be a very cheap but effective technique for this system. This method will unfortunately only work for specific systems where the virtual environment has fixed objects that can be built physically.

The collision detection for this system is not a standard technique, like using bounding boxes or bounding spheres. The back face removal technique was used to reduce the number of polygons that need to be checked for collisions. If a polygon is facing the tip of the gastroscope and too close to the tip, then the polygon is considered for warping. A warp detection distance is calculated when the system starts. It is used in the collision detection to recognise when a polygon is too close to the tip of the gastroscope. When the tip is too close to a polygon, the shape of the VR model must be deformed. Warping the

vertices of the polygons which are too close to the tip of the gastroscope performs this deformation.

3-D conditions can also be added to the VR model. Generating a grid and setting its y-coordinates to reconstruct the 3-D conditions added, proved to be very successful and effective. Even when warping 3-D conditions, the system is not very slow. The method developed for blending the 3-D conditions into the VR model using triangular polygon borders between a 3-D condition and the polygons of the VR model proved to be very successful.

For the system to be more useful, it provides an evaluation function. The trainer daemon evaluates what the student is doing when in Test Mode. An evaluation report can then be presented at the end of the exercise.

Chapter 8

Guided Tour of the System

8.1 Introduction

8.2 From the Instructor's Point of View

8.3 From the Trainee's Point of View

8.1 Introduction

This chapter presents an example of how the system would typically be used. The example will be explained from the instructor's point of view, as well as from the trainee's point of view. Please make sure to refer to all the mentioned colour plates (Appendix B) when reading sections 8.2 and 8.3. This chapter was written independently from chapters three to seven so that the reader can get an overall understanding of what the system does and how it works. It might also benefit the user to first look at movie clips "**\Movie Clips\4-Gastroscope in Physical Model.avi**" and "**\Movie Clips\VR System\4- Orientation and Organ Windows.avi**" on the accompanied CD-ROM to get an idea of how the system works. The first mentioned movie clip shows how the gastroscope is moved inside the physical model. The second video clip shows a computer-generated image that simulates that of a real stomach. It also shows a wireframe window that shows where the tip of the gastroscope is inside the VR model. Another window shows a rotating stomach which can be used to click on and the system will give information to the user as to which anatomical region he/she clicked on.

8.2 From the Instructor's Point of View

When the VR system is running, the instructor can set up two different kinds of tests, namely identification and/or navigational tests. Identification tests are tests set up by the instructor to teach a trainee basic identification skills, for example to identify an ulcer or polyp. Navigational tests are tests set up by the instructor to teach a trainee navigational skills, for example to take a biopsy. To set up a test for identification training, the instructor should make sure that the menu item **Identification skills,** under the **Train** menu item, is checked. See Plate 8.1. When the VR system starts, this menu item will be checked by default.

To set up an identification test, the instructor must add the abnormal conditions, which the trainee must identify, to the VR model. In this case it is a virtual model of a stomach. To add abnormal conditions to the virtual stomach, the virtual reality system's edit mode must be activated using the menu item **Edit**. A separate window, the condition browser, will appear. It will display all the abnormal conditions that could be inserted in the virtual stomach. The condition browser displays up to four conditions at a time, but only one condition is not displayed in wireframe. This condition is the currently selected one. A condition can be selected by using the scroll bar at the bottom, or selecting the condition from a list. If the instructor would, for example select an ulcer, the condition browser could look like Plate 8.2.

The selected condition can be placed anywhere inside the virtual stomach. Note that when in edit mode, the inside of the virtual stomach is not smooth any more, but very blocky. This makes it easier for the instructor to see where a condition can be placed. Using the gastroscope to move to a specific place in the virtual stomach, the instructor can use the left mouse button to click on the specific place inside the virtual stomach where the condition should be placed. Plate 8.3 illustrates an ulcer placed inside the virtual stomach.

Several conditions can be placed inside the virtual stomach. To deactivate edit mode the instructor should use the **Edit** menu item again. The inside of the stomach should now be smooth again. Refer to Plate 8.4.

The instructor has now set up a scenario inside the virtual stomach. This scenario can be saved using the **Save Identification Data** menu item, under the **File** menu item. Each condition's name and position is saved in this file and can be loaded again by the instructor for changes or by the trainee as a test.

To set up a test for navigational training, the instructor must make sure that the menu item **Navigation skills,** under the **Train** menu item, is checked. Plate 8.5 illustrates this.

The edit mode must be activated using the menu item **Edit**. A separate window with two combo boxes will appear with which one can set up certain navigational tasks. A navigational task has an *action* part and a *region* part on which the action should be performed, for example "Do biopsy on marked region". The "Do biopsy" part of the navigational task is the action part and the "on marked region" is the region on which the action should be performed. The two combo boxes are used to define a navigational task. Firstly the task action should be selected and then according to the selected action, certain regions are available to which the task applies. For example, to set up a navigational task to do a biopsy on a certain place inside the virtual stomach, select "Do biopsy on" from the first combo box. The second combo box will automatically show only one option, "Marked Region". This is shown by Plate 8.6.

Depending on the navigational task, the instructor should in some cases also specify where the marked region is inside the virtual stomach. This can be done by clicking with the left mouse button on a specific place inside the virtual stomach where for example a biopsy should be taken. This region will be marked with a white background and a blue cross on it. An example of a marked region can be seen in Plate 8.7. In Plate 8.7 the tip of the gastroscope is close to the VR model, therefore the marked region and polygons are shown quite prominently.

Several tasks can be constructed in this way, forming a list of navigational tasks that the trainee must perform. After the instructor has constructed the navigational tasks, the edit mode should be deactivated using the **Edit** menu item again. The inside of the virtual stomach should now be smooth again. Using the **Save Navigation Data** menu item, under the **File** menu item, this navigational task list could be saved like a test.

To summarise this section, the instructor can set up two different types of tests for a trainee. Identification tests can be used to improve basic identification skills and navigation tests to improve navigation skills with a gastroscope. The next section will explain how a trainee can use these tests to improve his/her skills.

8.3 From the Trainee's Point of View

A trainee can improve his/her basic identification and navigation skills using the tests which the instructor has set up. Like any other test, a trainee would want to first study or practise before the test. The virtual reality system therefore has two modes, *learn* mode and *test* mode. In learn mode, the trainee can practise identification and navigation skills by either loading old tests or setting up his/her own test. In learn mode the virtual reality system will give the trainee information about abnormal conditions. Extra information will also be given during navigational tasks to help the trainee. In test mode the trainee can be tested. The trainee is asked information about abnormal conditions. No extra information is given to the trainee during navigational tasks. The next paragraph will explain how a trainee can use the virtual reality system.

After the virtual reality system has been started, the trainee can load previously saved tests. To load a test for identification skills, the **Load Identification Data** menu item under the **File** menu can be used. As soon as a valid test is selected, the simulator will remove all current conditions from the virtual stomach and place all new conditions of the selected test in the virtual stomach. To load a test for navigational skills, the **Load Navigation Data** menu item under the **File** menu can be used. By default, the simulator uses the learn mode. To use the test mode, the trainee must be logged onto the simulator by using **Login**, under the **File** menu. This will ask for the trainee's name and a unique number, like a student number, to further identify the trainee. The current system does not have password protection for each user. Since this system is a training simulator, such a feature is important to add since the instructor would use evaluations of students. To switch between learn and test mode, the **Options** menu can be used.

If **Identification Skills**, under the **Train** menu, is checked, then the current test for identification training will be used. The trainee can now manoeuvre the gastroscope into the virtual stomach and examine the inside for certain abnormal conditions. Clicking on a condition with the left mouse button while in learn mode will show a description of the condition. Refer to Plate 8.8. Clicking on a condition with the left mouse button while in test mode will generate a multiple-choice question. Plate 8.9 illustrates such a multiple-choice question. The question must then be answered and the result will be reflected in an evaluation report.

If **Navigation Skills**, under the **Train** menu, is checked, then the current test for navigation training will be used. Details of the current task to be completed will be shown at the bottom of the main window. Plate 8.10 illustrates the message at the bottom of the main window to describe the navigation task that the trainee must complete. A biopsy tool reaching towards a marked region can also be seen.

The trainee can now manoeuvre the gastroscope inside the virtual stomach until the tip of the gastroscope is in the correct position to perform the requested task. If the system is in learn mode, the trainee can choose to repeat the current task or automatically move on to the next task. When the system is in test mode, a timer is used to time how long the trainee takes to perform the current task. This time is then displayed in an evaluation report. While performing the navigational task the timer is not shown to the trainee since in the case of doing a real procedure the trainee will not look at a timer either.

Using the **Evaluation** menu option, under the **File, User** menu, the trainee can get an evaluation report of the tests that have been completed. See Plate 8.11. Identification skills are evaluated by calculating the percentage of correct answers. Navigation skills are evaluated by showing how many tasks the trainee could complete and how long it took him/her. At this stage the virtual reality system does not store evaluation reports in a database. This means that as soon as the trainee quits from the virtual reality system, the evaluation report will be lost unless the evaluation report was printed. Just as the password feature for each student login, this feature is also an important one to add to the system.

To summarise this section, the trainee can use identification and navigation tests to improve his/her skills. The trainee can set up these tests in the same way that the instructor can or a previously saved test can be loaded.

The purpose of this chapter was to give the reader a broad overview of how the virtual reality system works from the end user's point of view. This will hopefully assist in better understanding sections in other chapters.

Chapter 9

System Tests and Evaluation

9.1 Introduction

9.2 Evaluation Aspects

9.2.1 The “Look Realistic” Aspect

9.2.2 The “Feel Realistic” Aspect

9.2.3 Speed of the System

9.2.4 Cost of the System

9.3 Technical Aspects

9.3.1 Minimum Requirements

9.3.2 Number of Vertices and Polygons in VR Model

9.3.3 Optimum Desktop Resolution

9.4 User Tests and Evaluation

9.4.1 The “Look Realistic” Aspect

9.4.2 The “Feel Realistic” Aspect

9.4.3 Speed of the System

9.4.4 Cost of the System

9.4.5 General Feedback

9.5 Conclusion

9.1 Introduction

This chapter will discuss the testing and evaluation of the system by medical experts. Note that all the tests discussed in this chapter were done using a computer meeting only the minimum requirements of the system, therefore a faster computer will give much better results. Section 9.3.1 discusses the minimum requirements for the system. A number of aspects are important for the success of the system. Therefore the tests and evaluation of the system will be based on these aspects. These aspects will be discussed in section 9.2 and are the following: how realistic the system *looks*, how realistic the system *feels*, the *speed* of the system and the *price* of the system. Technical aspects of the testing of the system will be discussed in section 9.3. Section 9.4 will discuss the evaluation of the system based on tests done by medical experts. The evaluation of the system in section 9.4 will be based on the aspects mentioned.

9.2 Evaluation Aspects

This section will discuss the aspects that are involved in evaluating this system. Section 9.4 will discuss the evaluation of the system based on these aspects.

9.2.1 The “Look Realistic” Aspect

The first aspect is whether the system generates images that *look* realistic. The following factors determine whether the images are realistic or not: the shape of the VR model, the images used to represent tissue inside the VR model, the representation of abnormal conditions in the VR model and the representation of therapeutic tools.

The shape of the VR model is determined by its vertices and polygons. If the VR model were modelled using too few vertices and polygons, the VR model would have a lot of flat surfaces, which would look unrealistic. The more vertices and polygons used, the more realistic the shape of the VR model would be, but if too many vertices and polygons were used, the system could be slowed down drastically so that the images generated are displayed too slowly. This will give a jerky effect and will not look smooth like the image of a real gastroscope. The number of vertices and polygons of the VR model for this system will be discussed further in section 9.3. Because the shape of the VR model

resembles the shape of the physical model, both the shape of the VR model and the shape of the physical model must look realistic.

Images that are used for texture maps on the polygons of the VR model play a very important role in making the generated images look realistic. Therefore real pictures of stomach tissue were scanned to get texture maps that look very realistic. Abnormal 3-D conditions also need to look very realistic inside the VR model. Therefore pictures of real abnormal conditions were used with the 3-D simulation of the 3-D abnormal conditions. Plates 6.3 to 6.5 show how a real image of an ulcer was used to obtain a texture map of an ulcer. The optimum desktop resolution (256 colours, 16-bit high colour, etc.) should also be used, otherwise the effect of using real pictures will be lost. See section 9.3 for more detail on the optimum desktop resolution.

Therapeutic tools must also be shown in the generated images, like a biopsy tool. These must be generated to resemble the real tool very closely, but care must be taken not to use too many vertices and polygons, since this could also slow down the system unnecessarily. The system should especially not display the generated images slowly when therapeutic tools are shown, because this will influence the hand-eye co-ordination of the trainee in a negative manner. Plate 7.5 shows how a biopsy tool was generated using standard objects like a cylinder and two half spheres.

9.2.2 The “Feel Realistic” Aspect

Since an important part of this system is to train navigation skills that have to do with hand-eye co-ordination training, the system should also *feel* realistic. Therefore a real gastroscope is used for the physical user interface. This is very important so that the trainee can also get used to the controls of the gastroscope and the “feel” of a real gastroscope. Movie clip “**\Movie Clips\4- Gastroscope in Physical Model.avi**” shows how the tip of the gastroscope is manoeuvred inside the physical model using the gastroscope controls.

When the system was tested the physical model of the stomach was made of fibreglass, which does not feel the same as a real stomach. The current physical model, which is

beyond the scope of this thesis, is made of silicon so that when the real gastroscope touches the physical model, it feels almost the same as touching the inside of a real stomach.

The more realistic the manoeuvring of the gastroscope inside the physical model feels, the more successful the system would be when training trainees.

9.2.3 Speed of the System

The aspect of speed was briefly addressed in section 9.2.1. This was thought to be a too important aspect not to elaborate on it more. Given that the system was developed to run on a PC (personal computer) and not a very fast computer like a Silicon Graphics computer, speed is a very important aspect.

The speed of the system is measured by the rate of frames per second, where each frame is a computer-generated image of the VR model from the viewpoint of the tip of the gastroscope. The speed with which a frame can be rendered is very dependent on the quality of the rendered image. Higher quality images will be generated slower than lower quality images. Looking at very popular computer games like “DOOM” and “Comanche”, it was noticed that most users would rather work with lower quality graphics and have more speed than the other way around. The faster these games were, the more realistic they felt, and therefore *very* high quality graphics are not always needed to ensure the success of a system. With this system speed is also more important than *very* high quality images. The trainee should not see or feel that the system is lagging behind the real gastroscope when moving the gastroscope around.

Most VR applications try to render more than between 30 frames per second, since it is in this range that the eye normally does not detect flickering any more [VIN95]. Generating images at a lower rate might therefore cause an application to be jerky.

9.2.1 Cost of the System

This system was specifically developed to run on a PC which costs much less than a graphics orientated computer, like a Silicon Graphics computer. The specifications of the PC on which the system was evaluated are given in section 9.3. These are also the minimum requirements for the system. The cost of the system includes a complete PC with the physical model, a gastroscope of which only the mechanical parts are working, a 3-D tracker and the software. A perfectly working gastroscope could also be used, but since this system only tracks the movements of the front tip, a broken gastroscope of which the mechanical parts are still working will suffice.

9.3 Technical Aspects

This section will discuss some technical aspects of the system before the user tests and evaluation are discussed. The following aspects will be discussed: The minimum requirements for a computer to run the system, the number of vertices and polygons in the VR model and the optimum desktop resolution. Note that all these aspects influence the speed of the system.

9.3.1 Minimum Requirements

This system was developed and tested on an Intel Pentium Pro 180 MHz with 64 Mbytes of RAM, with a S3 Trio video display adapter with 2 Mbytes of RAM. When the project was started, this was the top-of-the-line PC. By March 1999, the Pentium III was already released and very powerful 3-D graphic accelerator display cards were available. Since all the tests were done on the PC configuration mentioned, these were established as the minimum requirements for the system to work properly. Using newer PCs with 3-D accelerator display cards will obviously enhance the speed of the system.

9.3.2 Number of Vertices and Polygons in VR Model

The VR model consists of 2304 vertices and 576 polygons. A number of different VR models, for the same stomach, with different numbers of vertices and polygons were tried. It was decided to use this specific VR model because it could be rendered fast enough and

it looked very realistic. If less vertices and polygons are used, the system is faster and if more vertices and polygons are used, the system is slower.

9.3.3 Optimum Desktop Resolution

This section will explain how the optimum desktop resolution was established for this system by means of six tests. For the purpose of these tests 3-D conditions were generated, each made up of 53 vertices and 64 polygons. The frame rate of the simulator was evaluated as a combination of the number of 3-D conditions in the VR model as well as the colour depth of the desktop.

During these tests the width of the main window was 346 pixels and the height 354 pixels, which account for 122484 (346 x 354) pixels that have to be drawn per frame. If, for example one evaluation results in 10 frames per second, this will mean that 122484 x 10 = 1224840 pixels were drawn per second. The following are six tests done on the same PC with different configurations.

Test 1

Screen Resolution: 800x600
Colour Depth: 256 colour palette
Number of 3-D Conditions: 0
Frames per Second: 17.88

Test 2

Screen Resolution: 800x600
Colour Depth: 256 colour palette
Number of 3-D Conditions: 5
Frames per Second: 17.75

Test 3

Screen Resolution: 800x600

Colour Depth: 16-bit true colour

Number of 3-D Conditions: 0

Frames per Second: 12.14

Test 4

Screen Resolution: 800x600

Colour Depth: 16-bit true colour

Number of 3-D Conditions: 5

Frames per Second: 11.21

Test 5

Screen Resolution: 800x600

Colour Depth: 32-bit true colour

Number of 3-D Conditions: 0

Frames per Second: 8.45

Test 6

Screen Resolution: 800x600

Colour Depth: 32-bit true colour

Number of 3-D Conditions: 5

Frames per Second: 7.90

Reflecting on the above tests and referring to Plates 4.3 and 4.4 for the difference between using colour palettes or not, the most effective settings for this system are in test 3 and 4. The difference in quality between Plates 4.3 and 4.4 is very noticeable, but the difference in quality between a 16-bit high colour and a 32-bit true colour is not really noticeable to the human eye. Using 32-bit true colour makes the system much slower. Therefore to

maintain a good resolution for the textures and still have a high enough frames per second rendering rate, the 16-bit high colour setting was chosen.

9.4 User Evaluation

The user evaluation is based on expert reviews of the system. It is based on the aspects mentioned in section 9.2.

A prototype of this system was first introduced during the SAGES ’97 Congress (South African Gastroenterology Society), 30 May to 3 June 1997 in Bloemfontein, South Africa. The system was demonstrated to experts and tried out by experts. After that seven academic hospitals in South Africa were also visited and an improved prototype system was demonstrated to the gastroenterologists of each hospital. These academic hospitals are listed below. The field trials were in the form of one hour demonstrations which were followed by medical experts testing the system.

University of the Orange Free State
Universitas Hospital
Prof. H. Grundling
Dr. J.H. van Zyl

Stellenbosch University
Tygerberg Hospital
Dr. C.J. van Rensburg

University of Cape Town
Groote Schuur Hospital
Prof. P.C. Bornman
Dr. J.A. Louw

University of Natal
King Edward VIII Hospital
Prof. A.E. Simjee

Medical University of South Africa (MEDUNSA)

Garankuwa Hospital

Prof. C.F. van der Merwe

University of Pretoria

Pretoria Academic Hospital

Prof. S.K. Spies

Prof. J.H.R. Becker

Dr. I. Treadwell

University of the Witwatersrand

Baragwanath Hospital

Prof. I. Segal

9.4.1 The "Look Realistic" Aspect

During all the demonstrations, only one person, a trainee, complained that the computer-generated images do not look like a real stomach. Another person, a doctor, suggested that the simulated ulcer is not realistic at all. After showing him how the 3-D ulcer was generated using a photo of a real ulcer, he was satisfied with the realism. Plate 6.11 shows that abnormal conditions can be simulated very realistically. Most of the people did not have any problem with how realistic the computer-generated images looked, neither in terms of the shape of the VR model, nor the stomach tissue. Therefore it could be said that the evaluation shows that the "look realistic" aspect was successfully addressed.

9.4.2 The "Feel Realistic" Aspect

All the people agreed that using a real gastroscope is essential for such a simulator. Most people did not like the physical fibreglass model, since the force feedback is not realistic. If a rubbery model with realistic force feedback could be used, it would enhance the system a lot. Therefore the evaluation shows that the "feel realistic" aspect can be improved a lot by using a rubbery physical model.

9.4.3 Speed of the System

There were no complaints about the speed of the system, even if it only rendered between 11 and 12 frames per second. Therefore the evaluation shows that the aspect of “speed of the system” was successfully addressed. Note that these tests were done on the minimum computer required. The tests were done from a user’s point of view. If the users therefore did not complain about the aspect of speed, the speed of the system is adequate.

9.4.4 Cost of the System

When the price of the system was mentioned, it seemed that most doctors thought it to be very expensive. After explaining the alternative of having to buy a similar system that runs on a Silicon Graphics computer, which can cost up to five times that of a PC, they were convinced that the system is relatively inexpensive. Most of them also agreed that comparing the price of the system to that of a new relatively cheap gastroscope is not unreasonable. Therefore the evaluation shows that the price of the system is justified.

9.4.5 General Feedback

The idea of using such a training simulator was very well received in general. Most people were very impressed with the system and thought that it had great potential as a helping tool for trainees.

Most gastroenterologists asked if ERCP (Endoscopic Retrograde Cholangio Pancreatography) procedures could be trained using this system, since ERCPs are much more difficult than normal gastroenterology procedures. At that stage ERCPs could not be practised. It was therefore decided that the most important future improvements were to use a rubbery physical model with the system and to implement the practising of ERCPs.

9.5 Conclusion

Judging from the overall evaluation, this system can be a very valuable tool in aiding trainees to learn, understand and be trained much better than using only conventional ways of training. Most of the younger generation doctors were very excited about the simulator

and have admitted that such a simulator would have helped them a lot during their own training. The older generation doctors are more set in their ways and are a bit sceptical about the whole idea of using computer simulators for training. It must be stressed that this simulator is not intended to *replace* reality-based (conventional) training, but rather to *enhance* it by preparing the trainees for reality-based training.

This chapter discussed the testing and evaluation of the system by medical experts. It introduced the reader to the certain aspects that were identified on which evaluation of the system was based. Some technical aspects, like the minimum requirements, were also discussed. The user evaluation was discussed based on the evaluation aspects. The following chapter will summarise research done during the development of this whole system. It will also discuss possible future work on this system.

Chapter 10

Conclusion and Future Research

10.1 Introduction

Chapter 10 will discuss the highlights of this thesis, which involve the problems that had to be addressed and contributions that originated from their solutions. Advantages and disadvantages of this system, as well as future research will be discussed in sections 10.3 and 10.4. Section 10.5 will discuss the future of this system, as well as an evaluation of the system from the author's point of view.

10.2 Highlights of this Thesis

The main reason for the development of this training simulator was to have a virtual reality simulator, that runs on a PC, which can prepare medical trainees for reality-based (conventional) training of certain gastrointestinal endoscopic procedures. For such a simulator to be successful, the simulator has to *realistically* simulate what a trainee would *see* and *feel* when doing a real procedure. The following sections will discuss problems that had to be addressed during the development of this system and the contributions that originated from their solutions.

10.2.1 Problems Addressed

This section will discuss the main problems addressed during the development of this system. Their solutions will also be discussed.

The first problem that had to be addressed when the system was initiated was to obtain a computer graphic model of the stomach. The 3-D *shape* of this model had to be the same as a normal stomach. A 3-D computer graphic model could have been bought, or an expensive 3-D digitiser could have been used to obtain this model, but the computer graphic model was generated by digitising a moulded model of a stomach, using a 3-D tracker. The model could have been digitised in several ways. Design specifications were set up so that future models could be generated in the same way. One of the important design specifications is that each polygon in the VR model should be able to have its own rectangular image. Therefore the VR model consists only of polygons with four sides. The reason why polygons should be able to show an image is so that images of stomach tissue can be rendered onto the inside of the VR model. With this system the tip of the gastroscope could get very close to polygons and to ensure good image quality, each polygon needs to show its own texture of stomach tissue. To summarise this paragraph, a computer graphics model of the stomach was obtained using specific design specifications. The most important one is the specification that each polygon must be four sided to accommodate its own texture. The author's evaluation of these design specifications is that it was very successful. The only thing that could be a problem is that these design specifications would not be flexible enough to apply to more complex models, like a

model of the lungs, but complex models could be divided into a hierarchy of simple objects.

While studying pictures of the inside of different peoples' stomachs, it is noticeable that each stomach looks a bit different, especially the *colours*. A general picture was chosen to use as normal stomach tissue, to be displayed on each polygon in the VR model. This technique proved to be very successful. The texture of a real stomach could not be simulated by a hand-drawn picture. A picture of real stomach tissue had to be used. A real stomach has a lot of *folds* at certain places. These folds were ignored resulting in a VR model that is smooth on the inside. Users were still content with the visual results concerning the colours and the 3-D shape of the VR model. Therefore using pictures of real stomach tissue as textures for each polygon in the VR model proved to be very successful.

The simulation of abnormal *conditions* was very successfully implemented using 3-D conditions. First 3-D conditions have to be generated using the 3-D condition generator, by generating a flat 3-D grid and loading a processed image onto it. This grid can be manipulated using certain mathematical functions. Adding a condition inside the VR model is done by deleting the polygon/s where the condition should be added and then generating a grid similar to the 3-D condition in the 3-D hole left by deleting the polygon/s. One of the problems that became evident with the adding of conditions inside the VR model was that because the 3-D condition has a much higher density of polygons than the surrounding polygons of the VR model, the 3-D condition reflected much more light than the surrounding polygons. This made each condition stand out very prominently. To make the 3-D conditions blend more into the background of the VR model, the intensity and saturation of the polygons in the 3-D condition had to be set a little less. This was set by trial and error until the best results were obtained. The easiest way of obtaining satisfactory results was to set the intensity and saturation very low and then set it higher and higher until a good balance was obtained between the overall light reflection of the VR model and the 3-D conditions. The result was that the conditions blended into the VR model much better.

Therapeutic tools also had to be simulated. The more difficult procedures involve the manoeuvring of therapeutic tools and not just examining of the inside of the stomach. A biopsy tool and cauterisation tool were constructed using basic structures like cylinders and cones. The moving parts of the tools also had to be simulated, for example the moving parts of a biopsy tool. A lot of therapeutic tools exist, but only the two tools mentioned were constructed for simulation since these are the very basic tools. Within the scope of this study, real therapeutic tools were not used with the real gastroscope, but an extra mouse was used for moving the virtual tools forward and backward, and opening and closing it. Therefore practising to use a therapeutic tool did not feel realistic, but at least the tools looked the same as the real tools.[9]

For the simulator the "feel real" aspect is also simulated by the use of a real gastroscope to explore the inside of the physical model. In the early stages of the project the mouse was used to navigate inside the VR model. This was a very difficult and frustrating exercise. It was therefore decided that the use of a real instrument is essential for this simulator. Using a real gastroscope, a trainee can learn how to hold the instrument correctly and how to work its controls before he/she has to use it with a real patient. Force feedback is very important. If the trainee pushes the gastroscope against the side of the physical model, then he/she should feel it. An easy and cheap method of force feedback was implemented, called natural force feedback. The fibreglass physical model does not provide realistic force feedback. The current system uses a silicon physical model which does present a very realistic force feedback model.

When the trainee pushes the gastroscope against the side of the physical model, the tip of the gastroscope may not penetrate the VR model. A simple deforming technique had to be developed so that when the tip of the gastroscope gets too close to the side of the VR model and almost penetrates it, the vertices at that area is warped away from the tip of the gastroscope.

[9] The current system uses a real therapeutic tool with a micro-switch for opening and closing of the tool. The depth of how far the tool was pushed into the instrument channel is obtained from a second mouse roller.

10.3.2 Contributions

This section will discuss a few contributions that originated from the solutions of the problems discussed in the previous section.

The 3-D condition generator and the use of 3-D conditions in the VR model can be seen as contributions to research in medical VR systems. According to recent searches, no other medical VR simulator makes use of the technique of using 3-D abnormal conditions. This technique can be used to simulate abnormal conditions very accurately and a new technique was also developed to add the 3-D conditions to the VR model so that they successfully blend in between the polygons of the VR model. It is the author's opinion that these techniques can be used very successfully in other medical VR systems.

A simple method for deforming the shape of the VR model was developed, by using a method where the vertices of polygons that are almost touching the tip of the gastroscope, are warped away from the tip of the gastroscope in a certain direction. This warping method ensures that the tip of the gastroscope cannot penetrate the VR model. To determine if the tip of the gastroscope is almost touching a certain polygon, another method had to be developed which calculates the warp detection distance. This distance is used to determine if the tip of the gastroscope is too close to a certain polygon. The Free-Form Deformation (FFD) technique is an existing technique for deforming the shape of VR models. It works by surrounding an object with 3-D control points. The model can be deformed by moving some of the control values. If the Bernstein polynomials were associated with the control points, then three control values will result in a quadratic distribution and four values will result in a cubic distribution [VIN95]. The FFD technique could be used with this system, but it was decided to rather develop a more simple technique, since calculating quadratic or cubic results might be too time consuming. The new deforming method was implemented in the system and proved to be very successful.

10.3 Advantages and Disadvantages

The following two sections will discuss the advantages and disadvantages of using this system. The advantages of the system can almost be seen as contributions made to the field of gastroenterology, but will rather be discussed as advantages of using the system.

10.3.1 Advantages

This simulator can contribute to the field of gastroenterology in the following ways: Trainees can practise on it 24 hours a day. Therefore trainees do not have to wait like in reality-based training for his/her turn. If a trainee has problems with a certain manoeuvre then he/she can practise that specific manoeuvre on the model until he/she feels comfortable with it.

The virtual patient will not vomit, move or complain, which makes it much easier for a trainee to be introduced to a specialist field. Working with an “easy” patient will also help beginner trainees to gain confidence in the procedures.

Because the physical model is transparent, the trainee can actually see where the tip of the gastroscope is in orientation with the stomach and get a better idea of orientating the gastroscope inside the stomach.

Almost any abnormal condition can be placed in almost any place inside the stomach, which is very convenient for setting up different test scenarios. Conditions that occur very rarely can also be placed in the VR model so that trainees can be exposed to these conditions more frequently than in real life. Because it is a computer simulator, updating the simulator is not a big problem. The 3-D condition database can also be updated with different variants of the same condition.

Teaching the trainee how the controls of a gastroscope works before he/she has to work on a live patient, can also contribute to less wear and tear on the gastroscopes. Since these instruments are very expensive to service, this system could therefore prove to save hospitals a lot of money.

10.3.2 Disadvantages

Because it is not a real patient, trainees could learn to become impatient with certain techniques and handle the gastroscope more roughly than they should. In other words, bad habits may develop. That is why it is still very important for the instructor to be able to see what the trainee has done, making use of the video recorder function.

The force feedback of a fibreglass model is not at all the same as that of a real stomach. A rubbery physical model's force feedback will not be *exactly* the same as a real stomach, but it will be close enough so that a trainee will have some idea as to what a real procedure will feel like.

10.4 Future Research

This section will discuss improvements that can be made to the system and how other models could be incorporated with this system.

10.4.1 Improvements

Judging from the feedback from the field trials, the most important improvement should be the use of a rubber physical model rather than the solid fibreglass physical model. The ideal would be to have a rubber physical model with exactly the same elasticity as a human stomach to make the force feedback as realistic as possible. The problem is that the physical model should still be transparent. Experiments have already been done by 5DT <Fifth Dimension Technologies>, using rubber latex, which is transparent, but it is unfortunately not UV-resistant, so after a week or two the rubber is not transparent any more. 3M International is apparently doing research on a rubber that will be see-through and UV-resistant. Silicon might also be an option.[10]

Another very important function for this system should be to do ERCPs (Endoscopic Retrograde Cholangio Pancreatography). During an ERCP procedure, the ampulla of Vater has to be located which is situated in the duodenum, just behind the pylorus sphincter. The

[10] At the time of this writing the current system is using a silicon model. This was, however, regarded as beyond the scope of this study and is therefore not included in this thesis.

ampulla must then be entered with a cannulation tool in order to cannulate either the pancreatic duct or the bile duct. A radioactive liquid is injected into one of the ducts. The cannulation technique is very difficult and a simulator like this would be an excellent tool for trainees and even doctors to practice on.[11]

Trainees might find the use of a humanoid face and body on the physical and VR model helpful to know if the patient is lying on his/her left/right side or back, etc.

A larger database of 3-D conditions will also be required as well as a tool with which a person can generate his/her own 3-D conditions using new photos. These images have to be blended onto an image of normal tissue automatically. Currently the processing of these images are not done automatically.

The prototype was tested on a Pentium Pro 180 MHz. Using a faster computer with the new AGP (Accelerated Graphics Port) technology and a 3-D graphics accelerated card, more frames per second, maybe even more than twice as many, could be rendered. This could lead to putting in more detail in the VR model, like the stomach folds, which will make the system look even more realistic.

A more dynamic model could be implemented for simulating breathing of a patient. Adjusting the warping algorithm, the VR model can be set to periodically enlarge and return to its normal size. The problem would be to have a rubbery physical model that can be adjusted in correlation with the VR model.[12]

Unfortunately one of the things that cannot be simulated by the simulator at this stage is the gastroscope itself, meaning when turning the tip of the gastroscope back to look at where the gastroscope enters the stomach, no black pipe is seen. Simulating this has not been mentioned to ensure the success of the system and will therefore not be implemented in the near future.

[11] The current system already has the ERCP functionality, but it is seen as beyond the scope of this study.
[12] The current system already has a more dynamic VR model in which the pylorus sphincter opens and closes on regular intervals.

Other important features that need to be implemented are saving user's passwords in a database so that one user cannot perform tests using another user's name, and saving user evaluations in a database.

10.4.2 Other models

This simulator currently only allows for one general size stomach. The ideal would be to have different sizes of stomach models, since doing a gastroscopy on a child is different than doing it on an adult. Thinner instruments are used for gastroscopies on children.

Other body cavities can also be implemented using this system as a shell. The ideal would be to have a humanlike torso with different models. Other models could be for the following procedures:

Colonoscopy	- endoscopy in the colon
Bronchoscopy	- endoscopy in the lungs
Hysteroscopy	- endoscopy in the uterus
Cystoscopy	- endoscopy in the bladder

These models should be prepared in the same way the stomach has been prepared. Firstly the physical model should be built and digitised, then the VR model should be prepared, the region database should be created and finally the 3-D condition database should be created. Using these different sets of models would make it possible to perform most endoscopic procedures.

10.4.3 Models of other Organs or Body Cavities

Other models can also be implemented that do not necessarily use a flexible endoscope like with the endoscopic procedures described in the previous section. Other endocameras are much smaller and rigid. These models could be for knee surgery, surgery in nasal cavities and laparascopic procedures.

10.5 Future of this System

In this section the author will discuss his own evaluation of the system and thoughts on the future of this system.

Comparing this system to other medical VR simulators, this system can be seen as a great success. Searches have revealed no other medical VR simulators that run on a PC which also give the same image quality than this system. The use of natural force feedback also makes the system much cheaper. Combining the natural force feedback with real instruments provides the ideal training tool for medical trainees, because the system looks realistic and feels realistic.

Designing the system, the author (and developer of the system) from the beginning of the project kept in mind that the system has the potential to be used as a shell for other models too. Therefore the system design was carefully done to ensure the “upgrading” of the system by adding new models. It is difficult to evaluate the design of the system in this regard, since no other models have been added to the system yet. Evaluating the design specifications of the VR model, the author feels that the only thing that might be a problem is with complex models, like the lungs. The design specifications are flexible enough to use on a hierarchy of simple objects that form one complex model. This should therefore also not be a problem.

This system can prove to be a very useful training tool for medical trainees. Trainees can train at any time and learn faster, which makes the training costs cheaper. The current system is already much further developed and the improvements that had priority implemented. It will be interesting to see what other models, referred to in the previous section, will realise to work with this system. The system was developed for the company 5DT <Fifth Dimension Technologies>, and they plan to distribute it commercially. The system can be seen as being in a beta test phase at the moment. The next step would be to let selected academic hospitals work with the system for a certain time as beta testers before a final product would be ready for the market.

Bibliography

ALI97 Aliaga, Daniel G.
Virtual Objects in the Real World
Communications of the ACM, March 1997, Vol. 40, No. 3

AND96 Andrews, Dee H.
Edwards, Bernell J.
Mattoon, Joseph S.
Thurman, Richard A.
Potential Modeling and Simulation Contributions to Specialized Undergraduate Pilot Training
Education Technology, July 1996, p6

AMM92 Ammeraal, Leendert
Programming Principles in Computer Graphics
John Wiley & Sons, 1992

ASH70 Ashizawa, Shinroku
Kidokoro, Tsutomu
Endoscopic Color Atlas of Gastric Diseases
Bunkodo Co., Ltd., 1970

BAL96 Balaguer, Jean-Francis
Gobbetti, Enrico
3D User Interfaces for General-Purpose 3D Animation
Computer, August 1996, p71

BAY96 Bayarri, Salvador
Fernandez, Marcos
Perez, Mariano
Virtual Reality for Driving Simualtion
Communications of the ACM, May 1996, Vol. 39, No.5

BRO94 Brown-Wodaski, Donna
Wodaski, Ron
Virtual Reality Madness and More!
SAMS Publishing, 1994

BRY96 Bryson, Steve
Virtual Reality in Scientific Visualization
Communications of the ACM, May 1996, Vol. 39, No. 5

BUR93 Burdan, Richard L.
Faires, J.Douglas
Numerical Analysis
PWS Publishing Company, 1993

BUR94 Burdea, Grigore
Coiffet, Philippe
Virtual Reality Technology
John Wiley & Sons, Inc., 1994

COL97 Colin, Arnaud
Boire, Jean-Yves
A novel tool for rapid prototyping and development of simple 3D medical image processing applications on PCs.
Computer Methods and Programs in Biomedicine 53, 1997, 87-92

COM97a **Yamaha designs with Silicon Graphics**
Computer Graphics, August 1997, Vol. 8, No. 4, p7

COM97b **New Signs for Airport**
Computer Graphics, August 1997, Vol. 8, No. 4, p16

COM97c **Realtime 3D simulation to make Bay Bridge earthquake safe**
Computer Graphics, August 1997, Vol. 8, No. 4, p17

COM98 **Computer generated imagery is star of Lost in Space**
Computer Graphics, August 1998, Vol. 9, No. 4, p5

DEB97 De Berg, M.
Van Kreveld, M.
Overmars, M.
Schwarzkopf, O.
Computation Geometry
Springer, 1997

DEE96 Deering, Michael F.
The HoloSketch, VR Sketching System
Communications of the ACM, May 1996, Vol. 39, No. 5

ENC97 **Microsoft Encarta 97 Encyclopedia**
Microsoft Corporation

EXP98a **Virtual Reality helps clean up Chernobyl**
Expert Systems, February 1998, Vol. 15, No. 1, p66

EXP98b **VR goes to Mars, real-time**
Expert Systems, February 1998, Vol. 15, No. 1, p66

EXP98c **3D model creator**
Expert Systems, February 1998, Vol. 15, No. 1, p67

EXP98d **Wireless feature for VR gloves**
Expert Systems, February 1998, Vol. 15, No. 1, p67

FER96 Ferraro, Richard F.
Learn 3-D Graphics Programming on the PC
Addison-Wesley Developers Press, 1996

FOR93 Fordtran, John S.
Sleisenger, Marvin H.
Gastrointestinal Disease
W.B. Saunders Company, 1993

GRE96 Green, Mark
Halliday, Sean
A Geometric Modeling and Animation System for Virtual Reality
Communications of the ACM, May 1996, Vol. 39, No. 5

GRO97 **The 1997 Grolier Multimedia Encyclopedia**
Grolier Interactive Inc., 1997

GON93 Gonzalez, Rafael C.
Woods, Richard E.
Digital Image Processing
Addison-Wesley Publishing Company, 1993

GOO97 Goodrich, Michael T.
Tamassia, Roberto
Dynamic Ray Shooting and Shortest Path in Planar Subdivisions via Balanced Geodesic Triangulations
Journal of Algorithms 23, 1997, 51 - 73

HAM93 Hamit, Francis
Virtual Reality and the Exploration of Cyberspace
SAMS Publishing, 1993

HOD96 Hodges, Larry F.
Rothbaum, Barbara O.
Watson, Benjamin
Kessler, G. Drew,
Opdyke, Dan
A Virtual Airplane for Fear of Flying Therapy
Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium, March 1996

HOH96 Höhne, K.H.
Pflesser, Bernhard
Pommert, Andreas
Riemer, Martin
Schiemann, Thomas
Schubert, Rainer
Tiede, Ulf
A 'Virtual Body' Model for Surgical Education and Research
Computer, January 1996, p25

HOL96 Hollands, R. J.
Trowbridge, E. A.
A PC-based virtual reality arthroscopic surgical trainer
Proceedings Simulation in Synthetic Environments, New Orleans, 1996, p17-22

JAC94 Jacobson, Linda
Garage Virtual Reality
SAMS Publishing, 1994

JAM97 Jambon, Anne-Claire
Dubois, Patrick
Karpf, Sylvain
A Low-Cost Training Simulator for Initial Formation in Gynaecologic Laparoscopy
Proceedings of Computer Vision, Virtual Reality and Medical Robotics and Computer-Assisted Surgery, March 1997, p347

KOC97 Kochevar, Peter D.
Tecate and Interactive 3-D: Combining networking and 3-D objects
Dr. Dobb's Journal, July 1997, p72

LAN86 Lang, Serge
A First Course in Calculus
Springer-Verlag, 1986

LAR93 Larijani, L. Casey
The Virtual Reality Primer
McGraw-Hill, Inc., 1993

LUS97 Lussier, Kyle
Implicit Surface and Real-Time Graphics
Dr. Dobbs Journal, July 1997, No. 267

MAC98 Mace, Jeff
Mainstream Graphics: Database and spreadsheet data will be liberated from 2-D representation to 3-D charts that can be manipulated and viewed at all angles.
PC Magazine, June 9, 1998, p122

MAT96 Mattoon, Joseph S.
Modeling and Simulation: A Rationale for Implementing New Training Technologies
Education Technology, July 1996, p17

MER96 Meredith, William
Virtual Reality for Patients with Spinal Cord Injury
M.D. Computing, Vol. 13, No. 5, 1996

MES95 Meseure, Philippe
Rouland, Jean-Francois
Dubois, Patrick
Karpf, Sylvain
Chaillou, Christophe
A Retinal Laser Photocoagulation Simulator Overview
Proceedings of Computer Vision, Virtual Reality and Robotics in Medicine, April 1995, p105

OESGIF Olympys Endoscopy Systems Gastrointestinal Fiberscopes
Brochure from Olympus
Printed in Japan F387E-0394T

OZE98 Ozer, Jan
3-D Computing: Advanced instruction sets, memory and chip set acceleration will make computer images more lifelike than ever.
PC Magazine, June 9, 1998, p118

PAS98 Pascoe, Elaine
Virtual Reality Beyond the Looking Glass
Blackbirch Press, Inc., 1998

PAT97 Patel, Mayur

A Memory-Constrained Image-Processing Architecture

Dr. Dobb's Journal, July 1997, p24

POS96 Poston, Timothy

Serra, Louis

Dextrous Virtual Work

Communications of the ACM, May 1996, Vol. 39, No. 5

ROB96 Robb, Richard

Hanson, Dennis P.

Camp, Jon J.

Computer-Aided Surgery Planning and Rehearsal at Mayo Clinic

Computer, January 1996, p39

SAL90 Sales, S. L.

Hille, Einar

Calculus, One and Several Variables

John Wiley & Sons, 1990

SAT94 Satava, Richard M.

Emerging medical applications of virtual reality: A surgeon's perspective

Artificial Intelligence in Medicine 6, 1994, p281 - 288

SCH92 Scheiderman, Ben

Designing the User Interface

Addison-Wesley Publishing Company, 1992

SCH97 Schroeder, Will

Citriniti, Tom

Decimating Polygon Meshes

Dr. Dobb's Journal, July 1997, p109

SHI97 Shimizu, Shuichi
Numao, Masayuki
Constraint-based design for 3D shapes
Artificial Intelligence 91, 1997, 51 - 69

SIN96 Singh, Gurminder
Feiner, Steven K.
Thalmann, Daniel
Virtual Reality, Software and Technology
Communications of the ACM, May 1996, Vol. 39, No.5

STA97 Stanek, William R.
VRML Brings 3-D Worlds To the Web
PC Magazine, August 1997, p315

STU96 Studholme, C.
Hill, D.L.G.
Hawkes, D.J.
Automated 3-D registration of MR and CT images of the head
Medical Image Analysis, June 1996, Vol. 1, No. 2

STR88 Strang, Gilbert
Linear Algebra and its Applications
Harcourt Brace Javanovich, 1988

THO96 Thompson, Nigel
3-D Graphics Programming for Windows 95
Microsoft Press, 1996

VIN95 Vince, John
Virtual Reality Systems
Addison-Wesley Publishing Company, 1995

VON98 Von Schweber, Linda
Von Schweber, Erick
The Web's a 3-D World After All
PC Magazine, June 9, 1998, p45

VON98 Von Schweber, Linda
Von Schweber, Erick
Virtual Reality: Teams of workers will visually explore virtual prototypes together and interactively conduct simulations through VR technology.
PC Magazine, June 9, 1998, p186

VRN96a **VR in Medicine – Surgical Training**
VR News, April 1996, Vol. 5, Issue 5, p35

VRN96b **VR in Medicine – Virtual Endoscopy Training**
VR News, August 1996, Vol. 5, Issue 7, p25

VRN96c **VR in Medicine – Virtual Reality for Surgical Simulation**
VR News, August 1996, Vol. 5, Issue 7, p26

VRN96d **VR in Industrial Training; Application Supplement 2**
VR News, May 1996, Vol. 5, Issue 4, p.31

WAT96 Watt, Alan
Rendering Techniques: Past, Present and Future
ACM Computing Surveys, Vol. 28, No. 1, March 1996

WEI97 Weinhaus, Frederick M.
Devarajan, Venkat
Texture Mapping 3D Models of Real-World Scenes
ACM Computing, December 1997, Vol. 29, No. 4

WIL95 Wilson, John R.
Nichols, Sarah
Ramsey, Amanda
Virtual Reality Health and Safety: Facts, Speculation and Myths
VR News, November 1995, Vol. 4, Issue 9, p 20

YAG96 Yagel, Roni
Stredney, Don
Wiet, Gregory J.
Schmalbrock, Petra
Rosenberg, Louis
Sessanna, Dennis J.
Kurzion, Yair
King, Scott
Multisensory Platform for Surgical Simulation
Proceedings of IEEE 1996 Virtual Reality Annual International Symposium, 1996, p72

ZIE95 Ziegler, Rolf
Mueller, Wolfgang
Fischer, Georg
Dr. Goebel, Martin
A Virtual Reality Medical Training System
Proceedings of Computer Vision, Virtual Reality and Robotics in Medicine, April 1995, p282

Abstract:[13]

A Virtual Reality Gastrointestinal Trainer/Simulator has been developed to enable the simulation of gastrointestinal procedures on a personal computer (PC). Virtual Reality (VR) techniques are used in the construction of this computer-based system to enable the user to practice basic identification, navigational and therapeutical skills.

The system consists of a computer-based simulator, a 3-dimensional (3-D) tracker, an endoscope/endocamera and a life-size gastrointestinal model. A normal endoscope/endocamera is used with a hollow transparent life-size gastrointestinal model to provide maximum realism. The computer-based simulator contains a virtual 3-D model of the relevant gastrointestinal organ. Currently the stomach, esophagus and entry to the duodenum (upper G.I. region) are focused on. The position and orientation of the front tip of the endoscope/endocamera are tracked with the 3-D tracker. This data is relayed to the computer, which then calculates and displays the appropriate image on the computer screen as realistically as possible. The calculated image closely resembles the image which would be seen with a real endoscope/endocamera in a real patient. The image is continually updated in accordance with the movement of the endoscope/endocamera and the properties of the gastrointestinal model.

Some of the main problems that had to be addressed during the development of the system are: obtaining a 3-D computer graphic model of the stomach with the same shape, size, colour and texture as a real stomach; the simulation of abnormal conditions like ulcers, and how they can be placed inside the 3-D computer graphic model; the simulation of therapeutic tools, like biopsy foreceps; the implementation of realistic, but cheap force feedback; and the deforming of the 3-D computer graphic model when the user touches the inside of the stomach with the tip of the endoscope/endocamera.

The system is ideal for teaching, training, simulation, patient briefings and research. In this thesis the virtual reality system, its development and operation is described in detail.

[13] Parts of this abstract were submitted for a presentation at the SAGES ’97 Congress (South African Gastroenterology Society), 30 May to 3 June 1997 in Bloemfontein, South Africa, but were not accepted. The co-authors of these parts are Mr Paul Olckers, Dr. Johan P. du Plessis and Prof. C. Janse Tolmie.

Abstrak:

'n Skynwerklike ("virtual reality") gastrointestinale opleidings simulator is ontwikkel om gastrointestinale prosedures na te boots deur gebruik te maak van 'n persoonlike rekenaar. Skynwerklikheidstegnieke word gebruik met die ontwikkeling van hierdie rekenaargebaseerde stelsel sodat 'n gebruiker basiese identifikasie-, navigasie- en terapetiese tegnieke kan oefen.

Die stelsel bestaan uit 'n rekenaargebaseerde simulator, 'n drie-dimensionele (3-D) volger, 'n endoskoop/endokamera en 'n lewensgrootte gastrointestinale model. 'n Werklike endoskoop/endokamera word gebruik in 'n hol, deurskynende, lewensgrootte gastrointestinale model sodat die simulasie so realisties as moontlik kan voel. Die rekenaargebaseerde simulator gebruik 'n virtuele 3-D model van die relevante gastrointestinale orgaan. Hierdie studie fokus op die maag, esophagus en die ingang na die duodenum. Die posisie en orientasie van die punt van die endoskoop/endokamera word gevolg met die 3-D volger. Hierdie data word gegee aan die rekenaar wat dan die gepaste beeld so realisties as moontlik genereer en op die skerm vertoon. Die gegenereerde beeld lyk baie soos 'n beeld wat gesien sou word met 'n regte endoskoop/endokamera in 'n regte pasient. Die beeld word deurlopend opdateer volgens die beweging van die gastroskoop en die eienskappe van die gastrointestinale model.

Sommige van die probleme wat aangespreek moes word gedurende die ontwikkeling van die stelsel is: die generering van 'n 3-D rekenaar grafiese model van die maag met dieselfde vorm, grootte, kleur en tekstuur as 'n regte maag; die simulasie van abnormale kondisies soos maagsere, en hoe hullle in die 3-D rekenaar grafiese model geplaas kan word; die simulasie van terapetiese gereedskap, soos biopsie-tange; die implementasie van realistiese, maar goedkoop, krag terugvoer ("force feedback"); en die vormverandering van die 3-D rekenaar grafiese model wanneer die gebruiker die punt van die endoskoop/endokamera teen die wand van die maag laat raak.

Hierdie stelsel is ideaal vir onderwys, opleiding, simulasie, die inlig van pasiente en navorsing. Die skynwerklike stelsel, die ontwikkeling daarvan en die werking daarvan word in hierdie tesis beskryf.

Appendix A Pseudo Code

A.1 Generating the 3-D Grid

```
1. Set the grid's properties, like colour and shininess.
2. x = -1.0                    {generate vertices between –1 and 1}
3. y = 0.0                              {height is 0}
4. for CntX = 0 to GridSize-1           {horizontal}
 1. z = -1.0
 2. for CntZ = 0 to GridSize-1          {vertical}
  1. uv.u = CntX/(GridSize-1.0)         {for texture mapping}
  2. uv.v = CntZ/(GridSize-1.0)
  3. Create vertex using x, y, z, uv
  4. Add new vertex to grid
  5. z = z + 2/GridSize
 3. x = x + 2/GridSize
5. nr = 1                           {now create polygons}
6. for CntX = 1 to GridSize-1
 1. for CntZ = 1 to GridSize-1
  1. Create polygon using vertices with indexes: nr +1, nr, nr+GridSize,
                                                 nr+GridSize+1
  2. nr ++                          {row counter}
 2. nr++                            {column counter}
```

A.2 Applying mathematical functions to the 3-D Grid

```
1. if function Radius = 0
  1. a = Gradient / 0.001
   else
  2. a = Gradient / function Radius
2. middle_index = GridSize * (GridSize / 2 ) + 0.5
3. Get middle vertex, using middle_index
4. for cnt = 0 to NumVertices-1
  1. Get vertex, using cnt+1 as index
  2. Distance = sqrt ( (middle.x-vertex.x)² + (middle.z-vertex.z)² )
  3. if Distance <= distort Radius
    1. random = (rand() / MAXIMUM_RAND) * UserDestort
     else
    2. random = 0
  4. if Distance <= function Radius
    1. if f(x) is the "Butterworth" function
      1. fx = Dent / (1.0 + (Distance*a)² )
       else
      2. fx = Dent * sqrt(Radius² - Distance²) / Radius
    2. vertex.y = fx + random - (random/2.0)
     else
    3. vertex.y = 0
```

A.3 Normalising the VR Model

```
1. min.x = 3200              {ensures that a minimum value is obtained}
2. min.y = 3200              {from the first test value}
3. min.z = 3200
4. max.x = -3200             {ensures that a maximum value is obtained
5. max.y = -3200             {from the first test value}
6. max.z = -3200
7. num_vertices = Get number of vertices in model
8. for cnt = 0 to num_vertices-1    {Get min/max values for moving}
 1. Get vertex, using cnt+1 as index
 2. if vertex.x < min.x
  1. min.x = vertex.x
 3. if vertex.y < min.y
  1. min.y = vertex.y
 4. if vertex.z < min.z
  1. min.z = vertex.z
 5. if vertex.x > max.x
  1. max.x = vertex.x
 6. if vertex.y > max.y
  1. max.y = vertex.y
 7. if vertex.z > max.z
  1. max.z = vertex.z
9. for cnt = 0 to num_vertices-1          {move vertices to center}
 1. Get vertex, using cnt+1 as index
 2. centre.x = Div((min.x-max.x),2)-min.x
 3. centre.y = Div((min.y-max.y),2)-min.y
 4. centre.z = Div((min.z-max.z),2)-min.z
 5. vertex.x = vertex.x – centre.x
 6. vertex.y = vertex.y – centre.y
 7. vertex.z = vertex.z – centre.z
 8. Set vertex, using cnt+1 as index
10. min.x = 3200        {min/max values have now changed so recalculate}
11. min.y = 3200
12. min.z = 3200
```

```
13. max.x = -3200
14. max.y = -3200
15. max.z = -3200
16. for (cnt=0 to num_vertices-1) {Get min/max values for scaling}
 1. Get vertex, using cnt+1 as index
 2. if vertex.x < min.x
  1. min.x = vertex.x
 3. if vertex.y < min.y
  1. min.y = vertex.y
 4. if vertex.z < min.z
  1. min.z = vertex.z
 5. if vertex.x > max.x
  1. max.x = vertex.x
 6. if vertex.y > max.y
  1. max.y = vertex.y
 7. if vertex.z > max.z
  1. max.z = vertex.z
17. max_coordinate = max(max (max.x,max.y),max.z)
18. min_coordinate = min(min (min.x,min.y),min.z)
19. if max (Abs(max_coordinate),Abs(min_coordinate)) = 1
 1. return                                    {already normalised}
20. scaling_factor = max (Abs(max_coordinate),Abs(min_coordinate))
21. for cnt=0 to num_vertices-1
 1. Get vertex, using cnt+1 as index
 2. vertex.x = Div (vertex.x,scaling_factor)
 3. vertex.y = Div (vertex.y,scaling_factor)
 4. vertex.z = Div (vertex.z,scaling_factor)
 5. Set vertex at index cnt+1 to vertex
```

A.4 Preparing the VR Model for Smooth Shading

```
1. set normalized[0] .. normalized[num_vert-1] = -1
2. for cnt=0 to num_vert-1
 1. if (normalized[cnt] != 1)
  1. total.x = 0
  2. total.y = 0
  3. total.z = 0
  4. normal.x = 0
  5. normal.y = 0
  6. normal.z = 0
  7. number = 0
  8. Get cur_vertex, using cnt+1 as index
  9. for cnt2=0 to num_vert-1
   1. Get test_vertex, using cnt2+1 as index
   2. if (test_vert.x = cur_vert.x) AND
         (test_vert.y = cur_vert.y) AND
         (test_vert.z = cur_vert.z)
    1. Get cur_vertex normal vector in normal, using cnt2+1 as index
    2. total.x = total.x + normal.x
    3. total.y = total.y + normal.y
    4. total.z = total.z + normal.z
    5. number = number + 1
    6. normalized[cnt2] = 0              {found one to be normalized}
  10. Normalize the vector total
  11. for cnt2=0 to num_vert-1
   1. if normalized[cnt2] = 0
    1. Set vertex normal to vector total, using cnt2+1 as index
    2. normalized[cnt2] = 1
```

A.5 Calculation of the Warp Detection Distance

```
1. Warp Detection Distance = 0
2. For all polygons in the model do:
 1. Get polygon vertex indices in vertex
 2. middle_point[0].x = vertex[0].x + (vertex[1].x – vertex[0].x)/2
 3. middle_point[0].y = vertex[0].y + (vertex[1].y – vertex[0].y)/2
 4. middle_point[0].z = vertex[0].z + (vertex[1].z – vertex[0].z)/2
 5. middle_point[1].x = vertex[1].x + (vertex[2].x – vertex[1].x)/2
 6. middle_point[1].y = vertex[1].y + (vertex[2].y – vertex[1].y)/2
 7. middle_point[1].z = vertex[1].z + (vertex[2].z – vertex[1].z)/2
 8. middle_point[2].x = vertex[2].x + (vertex[3].x – vertex[2].x)/2
 9. middle_point[2].y = vertex[2].y + (vertex[3].y – vertex[2].y)/2
 10. middle_point[2].z = vertex[2].z + (vertex[3].z – vertex[2].z)/2
 11. middle_point[3]].x = vertex[3].x + (vertex[0].x – vertex[3].x)/2
 12. middle_point[3].y = vertex[3].y + (vertex[0].y – vertex[3].y)/2
 13. middle_point[3].z = vertex[3].z + (vertex[0].z – vertex[3].z)/2
 14 for cnt = 0 to 3 do
  1. for dist_cnt = 0 to 3 do
   1. Dist = sqrt (     (middle_point[cnt].x-vertex[dist_cnt].x)² +
                        (middle_point[cnt].y-vertex[dist_cnt].y)² +
                        (middle_point[cnt].z-vertex[dist_cnt].z)² ) / 2
   2. if Dist > Warp Detection Distance
    1. Warp Detection Distance = Dist
```

A.6 3-D Tracking Device

```
1. Read data from 3-D tracking device XPos, YPos, ZPos, XRot, YRot, Zrot
2. Initialise Matrix as a unit matrix
3. RotateMatrix (Matrix, 0,1,0, YRot , REPLACE)
4. RotateMatrix (Matrix, 1,0,0, XRot , PRECONCATENATE)
5. RotateMatrix (Matrix, 0,0,1, ZRot , PRECONCATENATE)
6. TranslateMatrix (Matrix,-(XPos+RegisterX)/STOMACH_SCALE_X,
                           -(YPos+RegisterY)/STOMACH_SCALE_Y,
                           -(ZPos+RegisterZ)/STOMACH_SCALE_Z,
                                            POSTCONCATENATE)
```

A.7 Collision Detection

```
1. Min_Distance = 32000       {ensures the first test value is minimum}
2. ListPos = 0
3. Get Camera look at direction in camera_dir
4. Normalize camera_dir
5. Get Camera position in camera_pos
6. For all polygons in model do:
 1. Get polygon normal vector
 2. Dot = DotProduct (polygon normal, camera_dir)
 3. if Dot < 0.20   {then camera faces polygon}
  1. Get polygon center
  2. Distance = sqrt ( (camera_pos.x-center.x)² +
                       (camera_pos.y-center.y)² +
                       (camera_pos.z-center.z)² )
  3. if Distance < Min_Distance
   1. Min_Distance = Distance
   2. Closest Polygon = current polygon
  4. if Distance <= WarpRadius  {then add polygon to warp list}
   1. PolygonWarpList[ListPos] = current polygon
```

```
   2. ListPos = ListPos + 1
```

A.8 Warping

```
1. Get direction in which Camera looks at in vector: direction
2. Normalize direction
3. Get Camera position in cam_pos
4. for cnt = 0 to ListPos-1
  1. Get vertex indices of polygon stored at PolygonWarpList[cnt]
     in vertex_index    {get all indices of vertices in this polygon}
  2. for cnt2 = 0 to 3        {always 4 sided polygon}
   1. if VertexData[vertex_index[cnt2]-1] not warped  {then warp it}
    1. Get vertex, using vertex_index[cnt2] as index
    2. Get distance to camera position
     1. Distance = sqrt((cam_pos.x-vertex.x)² +
                        (cam_pos.y-vertex.y)² +
                        (cam_pos.z-vertex.z)² )
    3. if Distance <=  WarpRadius
     1. direction = SubtractVector (vertex,cam_pos)
     2. Normalize (direction)
     3. Dot = DotProduct (direction,
                          VertexData[vertex_index[cnt2]-1].Normal)
     4. if Dot < 0.0 then                  {they look at each other}
      1. A.x = -VertexData[v_index[cnt2]-1].Normal.x
      2. A.y = -VertexData[v_index[cnt2]-1].Normal.y
      3. A.z = -VertexData[v_index[cnt2]-1].Normal.z
      4. B.x = cam_pos.x-vertex.x
      5. B.y = cam_pos.y-vertex.y
      6. B.z = cam_pos.z-vertex.z
      7. TopEq = A.x*B.x + A.y*B.y + A.z*B.z
      8. BottomEq = A.x*A.x + A.y*A.y + A.z*A.z
      9. P.x = TopEq *A.x / BottomEq
      10. P.y = TopEq *A.y / BottomEq
      11. P.z = TopEq *A.z / BottomEq
      12. BP = sqrt ( (B.x-P.x)² +
                      (B.y-P.y)² +
                      (B.z-P.z)² )
      13. if BP <= Warp Detection Distance then       {use Pythagorus}
        1. QP = sqrt ((Warp Detection Distance)² - (BP)²)
        2. Q.x = P.x + A.x * QP
        3. Q.y = P.y + A.y * QP
        4. Q.z = P.z + A.z * QP
        5. AQ = sqrt ( (Q.x)² + (Q.y)² + (Q.z)² )
      14. else
        1. AQ = 0
     5. else
      1. AQ = 0
     6. if AQ <> 0 then                    {warp vertex using AQ}
      1. vertex.x = vertex.x + A.x * AQ
      2. vertex.y = vertex.y + A.y * AQ
      3. vertex.z = vertex.z + A.z * AQ
      4. VertexData[vertex_index[cnt2]-1].Warped = True
```

A.9 How the available polygons are generated

```
1. Initialize PolygonGrid, a 13x13 array with zeros
2. PolygonGrid[6][6] = selected polygon's integer tag
{Now Search Upper Left Quadrant for unavailable grid positions}
3. for cnt = 0 to 5
 1. current_poly = PolygonGrid[6-cnt][6-cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6-cnt to 1                        {search to left}
   1. if PolygonGrid[cnt2-1][6-cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon left of current_poly
    3. if current_poly = -1              {set all to left to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to cnt2-1
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1        {unavailable}
    4. else
     1. PolygonGrid[cnt2-1][6-cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6-cnt][6-cnt]
  3. for cnt2 = 6-cnt to 1                          {search up}
   1. if PolygonGrid[6-cnt][cnt2-1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on top of current_poly
    3. if current_poly == -1             {set all upwards to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to 12
      1. for gridy = 0 to cnt2-1
       1. PolygonGrid[gridx][gridy] = -1        {unavailable}
    4. else
     1. PolygonGrid[6-cnt][cnt2-1] = current_poly     {available}
  4. if PolygonGrid[6-cnt-1][6-cnt-1] <> -1
   1. PolygonGrid[6-cnt-1][6-cnt-1] = polygon in the upper left
          corner of PolygonGrid[6-cnt][6-cnt]         {available}
   2. if PolygonGrid[6-cnt-1][6-cnt-1] == -1
    1. for gridx = 0 to 6-cnt-1
     1. for gridy = 0 to 6-cnt-1
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    2. cnt = 6  (finished with quadrant)
{Now Search Lower Left Quadrant for unavailable grid positions}
4. for cnt = 0 to 5
 1. current_poly = PolygonGrid[6-cnt][6+cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6-cnt to 1               {search to the left}
   1. if PolygonGrid[cnt2-1][6+cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon left of current_poly
    3. if current_poly = -1              {set all to left to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to cnt2-1
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1             {unavailable}
    4. else
     1. PolygonGrid[cnt2-1][6+cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6-cnt][6+cnt]
  3. for cnt2 = 6+cnt to 11                         {search Down}
   1. if PolygonGrid[6-cnt][cnt2+1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on underneath current_poly
    3. if current_poly =-1               {set all downward to –1}
     1. current_poly = old_poly
```

```
     2. for gridx = 0 to 12
      1.for gridy = cnt2+1 to 12
       1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[6-cnt][cnt2+1] = current_poly     {available}
  4. if (PolygonGrid[6-cnt-1][6+cnt+1] != -1)
   1. PolygonGrid[6-cnt-1][6+cnt+1] = polygon in the lower left
         corner of PolygonGrid[6-cnt][6+cnt]
   2. if PolygonGrid[6-cnt-1][6+cnt+1] == -1
    1. for gridx = 0 to 6-cnt-1
     1. for gridy = 6+cnt+1 to 12
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    2. cnt = 6 (finished with quadrant)
 3. PolygonGrid[0][12] = -1
{Search Lower Right Quadrant for unavailable grid positions}
5.for cnt = 0 to 6
 1. current_poly = PolygonGrid[6+cnt][6+cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6+cnt to 11                       {search to the right}
   1. if PolygonGrid[cnt2+1][6+cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon to the right of current_poly
    3. if current_poly =-1                {set all to right to –1}
     1. current_poly = old_poly
     2. for gridx = cnt2+1 to 12
      1. for gridy = 0 to 12
       1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[cnt2+1][6+cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6+cnt][6+cnt]
  3. for cnt2 = 6+cnt to 11                      {search down}
   1. if PolygonGrid[6+cnt][cnt2+1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on underneath current_poly
    3. if current_poly = -1               {set all downwards to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to 12
      1. for gridy = cnt2+1 to 12
       1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[6+cnt][cnt2+1] = current_poly     {available}
  4. if PolygonGrid[6+cnt+1][5+cnt+1] <> -1
   1. PolygonGrid[6+cnt+1][6+cnt+1] = polygon in the lower right
                                corner of PolygonGrid[6+cnt][6+cnt]
   2. if PolygonGrid[6+cnt+1][6+cnt+1] = -1
    1. for gridx = 6+cnt+1 to 12
     1. for gridy = 6+cnt+1 to 12
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    2. cnt = 6                                    {finished with quadrant}
{Now Search Upper Right Quadrant for unavailable grid positions}
6. for cnt = 0 to 5
 1. current_poly = PolygonGrid[6+cnt][6-cnt]
 2. if current_poly <> -1
  1. for cnt2 = 6+cnt to 11               {search to the right}
   1. if PolygonGrid[cnt2+1][6-cnt] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon to the right of current_poly
    3. if current_poly =-1                       {set all to right to –1}
     1. current_poly = old_poly
     2. for gridx = cnt2+1 to 11
      1. for gridy = 0 to 12
```

```
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[cnt2+1][6-cnt] = current_poly     {available}
  2. current_poly = PolygonGrid[6+cnt][6-cnt]
  3. for cnt2=6-cnt to 1                              {search up}
   1. if PolygonGrid[6+cnt][cnt2-1] <> -1
    1. old_poly = current_poly
    2. current_poly = polygon on top of current_poly
    3. if current_poly =-1                      {set all upward to –1}
     1. current_poly = old_poly
     2. for gridx = 0 to 12
      1. for gridy = 0 to cnt2-1
       1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    4. else
     1. PolygonGrid[6+cnt][cnt2-1] = current_poly     {available}
  4. if PolygonGrid[6+cnt+1][6-cnt-1] <> -1
   1. PolygonGrid[6+cnt+1][6-cnt-1] = polygon in the upper right
                                corner of PolygonGrid[6+cnt][6-cnt]
   3. if PolygonGrid[6+cnt+1][6-cnt-1] = -1
    1. for gridx = 6+cnt+1 to 12
     1. for gridy = 0 to 6-cnt-1
      1. PolygonGrid[gridx][gridy] = -1              {unavailable}
    2. cnt = 6                              {finished with quadrant}
 3. PolygonGrid[12][0] = -1
```

A.10 How the 3-D conditions are fitted into the VR model

```
1. Calculate original 3-D condition surface for scaling by:
 1. Get LL lower left vertex of condition
 2. Get UL upper left vertex of condition
 3. Get UR upper right vertex of condition
 4. Get LR lower right vertex of condition
 5. original_surface = CalcTriangleSurface(UL,LL,LR) +
                       CalcTriangleSurface(UL,UR,LR)
2. Calculate surface of polygon to be replaced by condition by:
 1. Get LL lower left vertex of polygon
 2. Get UL upper left vertex of polygon
 3. Get UR upper right vertex of polygon
 4. Get LR lower right vertex of polygon
 5. poly_surface = CalcTriangleSurface(UL,LL,LR) +
                   CalcTriangleSurface(UL,UR,LR)
3. Calculate scaling factor for fitted condition’s grid heights by:
 1. scaling_factor = poly_surface/original_surface
4. Get normal vectors of polygon vertices by:
 1. Get LL_normal lower left vertex of polygon
 2. Get UL_normal upper left vertex of polygon
 3. Get UR_normal upper right vertex of polygon
 4. Get LR_normal lower right vertex of polygon
5. Calculate direction vectors for each side of polygon by:
 1. Left_Dir = SubtractVector (UL,LL)
 2. Right_Dir = SubtractVector (UR,LR)
 3. Upper_Dir = SubtractVector (UR,UL)
 4. Lower_Dir = SubtractVector (LR,LL)
 5. Normalize (Left_Dir)
 6. Normalize (Right_Dir)
 7. Normalize (Lower_Dir)
 8. Normalize (Upper_Dir)
6. Calculate distance for upper and lower sides by:
 1. Upper_Dist = sqrt ( (UR.x-UL.x)² + (UR.y-UL.y)² + (UR.z-UL.z)² )
```

```
 2. Lower_Dist = sqrt ( (LR.x-LL.x)² + (LR.y-LL.y)² + (LR.z-LL.z)² )
7. Calculate Border points LLB and ULB by:(Lower Left and Upper Left)
 1.Get Lower point
  1. LPoint.x = LL.x + Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.x
  2. LPoint.y = LL.y + Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.y
  3. LPoint.z = LL.z + Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.z
 2. Get Upper point
  1. UPoint.x = UL.x + Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.x
  2. UPoint.y = UL.y + Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.y
  3. UPoint.z = UL.z + Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.z
 3. Calculate direction vector between the upper and lower points
  1. Left_Dir = SubtractVector (UPoint,LPoint)
  2. Normalize (Left_Dir)
 4. Calculate distance for left side border points
  1. Left_Dist = sqrt ( (UPoint.x-LPoint.x)² +
                        (UPoint.y-LPoint.y)² +
                        (UPoint.z-LPoint.z)² )
 5. LLB.x = LPoint.x + Left_Dist * COND3D_BORDER / 200 * Left_Dir.x
 6. LLB.y = LPoint.y + Left_Dist * COND3D_BORDER / 200 * Left_Dir.y
 7. LLB.z = LPoint.z + Left_Dist * COND3D_BORDER / 200 * Left_Dir.z
 8. ULB.x = UPoint.x - Left_Dist * COND3D_BORDER / 200 * Left_Dir.x
 9. ULB.y = UPoint.y - Left_Dist * COND3D_BORDER / 200 * Left_Dir.y
 10. ULB.z = UPoint.z - Left_Dist * COND3D_BORDER / 200 * Left_Dir.z
8. Calculate Border points, LRB,URB by:       (Lower Right, Upper Right)
 1. Get Lower point
  1. LPoint.x = LR.x - Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.x
  2. LPoint.y = LR.y - Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.y
  3. LPoint.z = LR.z - Lower_Dist * COND3D_BORDER / 200 * Lower_Dir.z
 2. Get Upper point
  1. UPoint.x = UR.x - Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.x
  2. UPoint.y = UR.y - Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.y
  3. UPoint.z = UR.z - Upper_Dist * COND3D_BORDER / 200 * Upper_Dir.z
 3. Calculate direction vector between the upper and lower points by:
  1. Right_Dir = SubtractVector (UPoint,LPoint)
  2. Normalize (Right_Dir)
 4. Calculate distance between upper and lower points by:
  1. Right_Dist = sqrt ( (UPoint.x-LPoint.x)² +
                         (UPoint.y-LPoint.y)² +
                         (UPoint.z-LPoint.z)² )
 5. LRB.x = LPoint.x + Right_Dist * COND3D_BORDER / 200 * Right_Dir.x
 6. LRB.y = LPoint.y + Right_Dist * COND3D_BORDER / 100 * Right_Dir.y
 7. LRB.z = LPoint.z + Right_Dist * COND3D_BORDER / 100 * Right_Dir.z
 8. URB.x = UPoint.x - Right_Dist * COND3D_BORDER / 100 * Right_Dir.x
 9. URB.y = UPoint.y - Right_Dist * COND3D_BORDER / 100 * Right_Dir.y
 10. URB.z =UPoint.z - Right_Dist * COND3D_BORDER / 100 * Right_Dir.z
9. Upper_Step = sqrt ( (URB.x-ULB.x)² +
                       (URB.y-ULB.y)² +
                       (URB.z-ULB.z)² ) / (URx-LLx)
10. Lower_Step = sqrt ( (LRB.x-LLB.x)² +
                        (LRB.y-LLB.y)² +
                        (LRB.z-LLB.z)² ) / (URx-LLx)
11. Transfor condition grid points into polygon space by:
 1. for cnt = 0 to URx-LLx                              (horizontal)
  1. Get Lower point by:
   1. LPoint.x = LLB.x + (cnt * Lower_Step * Lower_Dir.x)
   2. LPoint.y = LLB.y + (cnt * Lower_Step * Lower_Dir.y)
   3. LPoint.z = LLB.z + (cnt * Lower_Step * Lower_Dir.z)
  2. Get Upper point by:
   1. UPoint.x = ULB.x + (cnt * Upper_Step * Upper_Dir.x)
   2. UPoint.y = ULB.y + (cnt * Upper_Step * Upper_Dir.y)
   3. UPoint.z = ULB.z + (cnt * Upper_Step * Upper_Dir.z)
```

```
   3. Calculate direction vector between the upper and lower by:
    1. Left_Dir = SubtractVector (UPoint,LPoint)
    2. Normalize (Left_Dir)
   4. Calculate distance and step distance for each side by:
    1. Left_Step = sqrt ( (UPoint.x-LPoint.x)² +
                          (UPoint.y-LPoint.y)² +
                          (UPoint.z-LPoint.z)² ) / (URy-LLy)
   5. for cnt2 = 0 to URy-Lly                              (vertical)
    1. Calculate flat grid by:
     1. grid[cnt][cnt2].x = LPoint.x + (cnt2 * Left_Step * Left_Dir.x)
     2. grid[cnt][cnt2].y = LPoint.y + (cnt2 * Left_Step * Left_Dir.y)
     3. grid[cnt][cnt2].z = LPoint.z + (cnt2 * Left_Step * Left_Dir.z)
    2. Add the height of each grid point by:
     1. vert_cnt = (cnt * grid_size) + cnt2
     2. Get original grid height of vertex with index vert_cnt+1
     3. Height = original_height * scaling_factor
     4. grid[cnt][cnt2].x = grid[cnt][cnt2].x + (Height *avg_normal.x)
     5. grid[cnt][cnt2].y = grid[cnt][cnt2].y + (Height *avg_normal.y)
     6. grid[cnt][cnt2].z = grid[cnt][cnt2].z + (Height *avg_normal.z)
    3. Add texture map info by:
     1. U[cnt][cnt2] = ( cnt / (grid_size-1)) *
                       ( (100-COND3D_BORDER)/100) +
                       (COND3D_BORDER / 100)
     2. V[cnt][cnt2] = ( cnt2 / (grid_size-1)) *
                       ( (100-COND3D_BORDER)/100) +
12. Add UL,LL, UR and LR vertices of polygon to condition
13. Set LL texture map info to 0,0
14. Set UL texture map info to 0,1
15. Set UR texture map info to 1,1
16. Set LR texture map info to 1,0
17. Add left border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor-2
  1. triangle_vertices[0] = LL_Index
  2. triangle_vertices[1] = cnt+1
  3. triangle_vertices[2] = cnt+2
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. triangle_vertices[0] = LL_Index
 4. triangle_vertices[1] = factor
 5. triangle_vertices[2] = UL_Index
 6. Add new polygon, using triangle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = UL_Index
  2. triangle_vertices[1] = cnt+1
  3. triangle_vertices[2] = cnt+2
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
18. Add top border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor-2
  1. triangle_vertices[0] = UL_Index
  2. triangle_vertices[1] = (cnt*grid_size)+ grid_size
  3. triangle_vertices[2] = (cnt+1)*(grid_size) + grid_size
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. vertices[0] =UL_Index
 4. vertices[1] =((LLx+factor-1)*(grid_size))+ URy+1 //-1+1
 5. vertices[2] =UR_Index
 6. Add new polygon, using triangle_vertices
```

```
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = UR_Index
  2. triangle_vertices[1] =(cnt*grid_size) + grid_size
  3. triagnle_vertices[2] = (cnt+1)*grid_size + grid_size
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
19. Add right border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor-2
  1. triangle_vertices[0] = UR_Index
  2. triangle_vertices[1] =((grid_size-1)*grid_size)+ grid_size-cnt
  3. triangle_vertices[2] =((grid_size-1)*grid_size)+ grid_size-1-cnt
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. triangle_vertices[0] = UR_Index
 4. triangle_vertices[1]=((grid_size-1)*grid_size)+grid_size-factor-1
 5. triangle_vertices[2] = LR_Index
 6. Add new polygon, using triangle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = LR_Index
  2. triangle_vertices[1] =((grid_size-1)*grid_size)+ grid_size-cnt
  3. triangle_vertices[2] =((grid_size-1)*grid_size)+ grid_size-1-cnt
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
20. Add bottom border triangles by:
 1. factor = grid_size/2.0
 2. for cnt = 0 to factor - 2
  1. triangle_vertices[0] = LR_Index
  2. triangle_vertices[1] =((grid_size-1-cnt)*grid_size) + 1
  3. triangle_vertices[2] =((grid_size-cnt-2)*grid_size) + 1
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
 3. triangle_vertices[0] = LR_Index
 4. triangle_vertices[1] =((grid_size-factor-2)*grid_size) + 1
 5. triangle_vertices[2] = LL_Index
 6. Add new polygon, using tringle_vertices
 7. Set triangle texture same as condition texture
 8. for cnt = factor-1 to grid_size-2
  1. triangle_vertices[0] = LL_Index
  2. triangle_vertices[1] =((grid_size-cnt)*grid_size) + 1
  3. triangle_vertices[2] =((grid_size-cnt-2)*grid_size) + 1
  4. Add new polygon, using triangle_vertices
  5. Set triangle texture same as condition texture
```

A.11 Calculate a 3-D Triangle's Surface

```
1. Translate triangle to origin by:
 1. B = SubtractVector (b,c)
 2. A = SubtractVector (a,c)
2. Calculate a point P which lies on the base vector A
    so that vector BP is orthogonal on A
 1. P.x = (A.x*B.x + A.y*B.y + A.z*B.z)*A.x/
          (A.x*A.x + A.y*A.y + A.z*A.z)
 2. P.y = (A.x*B.x + A.y*B.y + A.z*B.z)*A.y/
          (A.x*A.x + A.y*A.y + A.z*A.z)
 3. P.z = (A.x*B.x + A.y*B.y + A.z*B.z)*A.z/
          (A.x*A.x + A.y*A.y + A.z*A.z)
```

3. Calculate triangle Height by:
 1. Height = sqrt ($(P.x-B.x)^2$ +
 $(P.y-B.y)^2$ +
 $(P.z-B.z)^2$)
4. Calculate triangle Base by:
 1. Base = sqrt ($(A.x)^2$ + $(A.y)^2$ + $(A.z)^2$)
5. Surface = 0.5*Base*Height

Appendix B Colour Plates

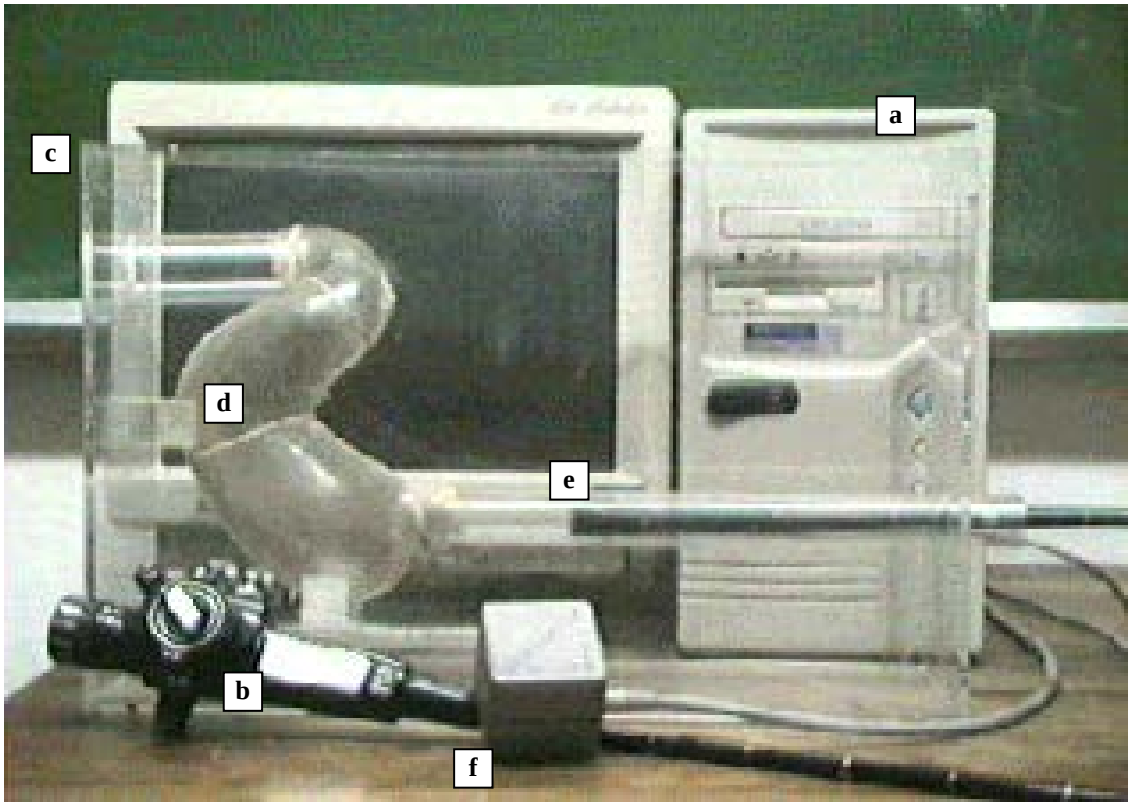

Plate 1.1 – Illustrates the components of the VR system. The components are the following: a) Computer, b) Standard Gastroscope, c) Perspex Box, d) Model of Stomach, e) 3-D Tracker Receiver, f) 3-D Tracker Transmitter. Also refer to Figure 1.1.

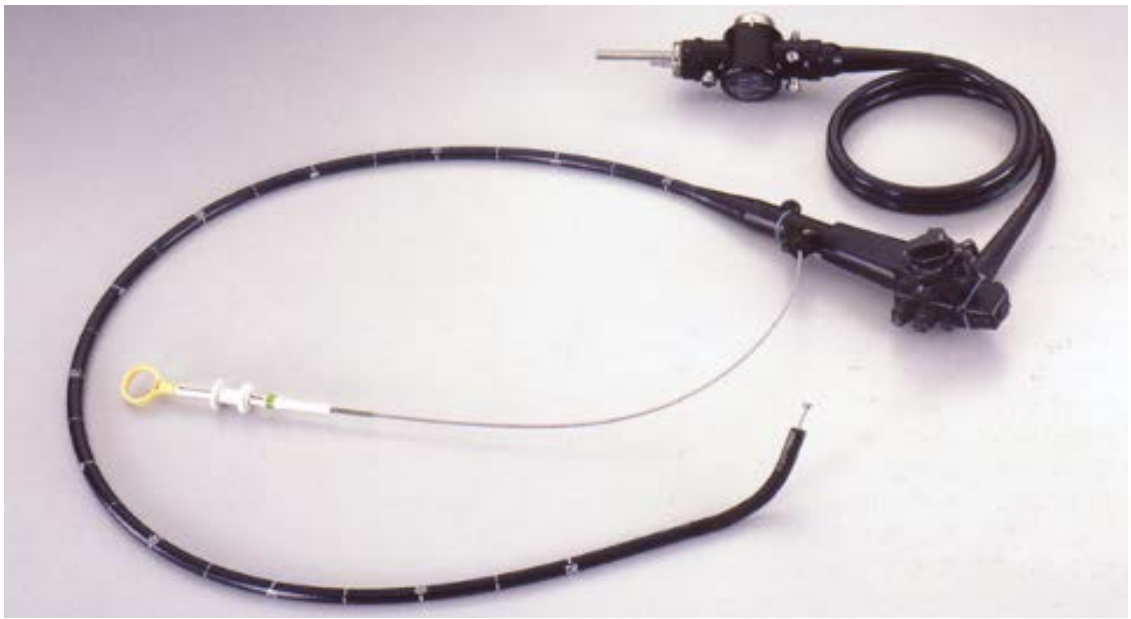

Plate 2.1 – An example of a gastroscope with therapeutic tool (Courtesy: Olympus).

Plate 2.2 – Shows how a gastroscope is held (Courtesy: Olympus).

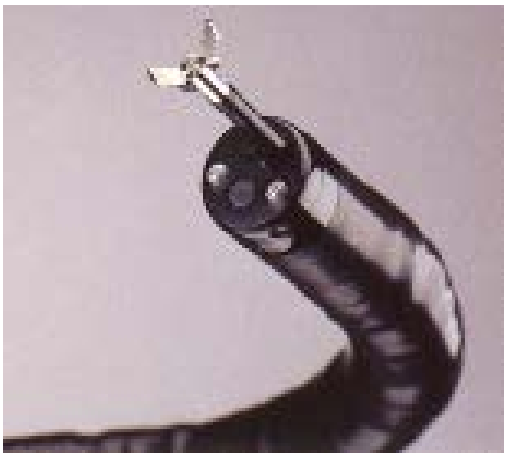

Plate 2.3 – Magnification of the front tip of a gastroscope with biopsy tool (Courtesy: Olympus).

Plate 2.4 – Shows the 3-D tracker receiver on the left hand side attached to the tip of a gastroscope and the 3-D tracker transmitter on the right hand side.

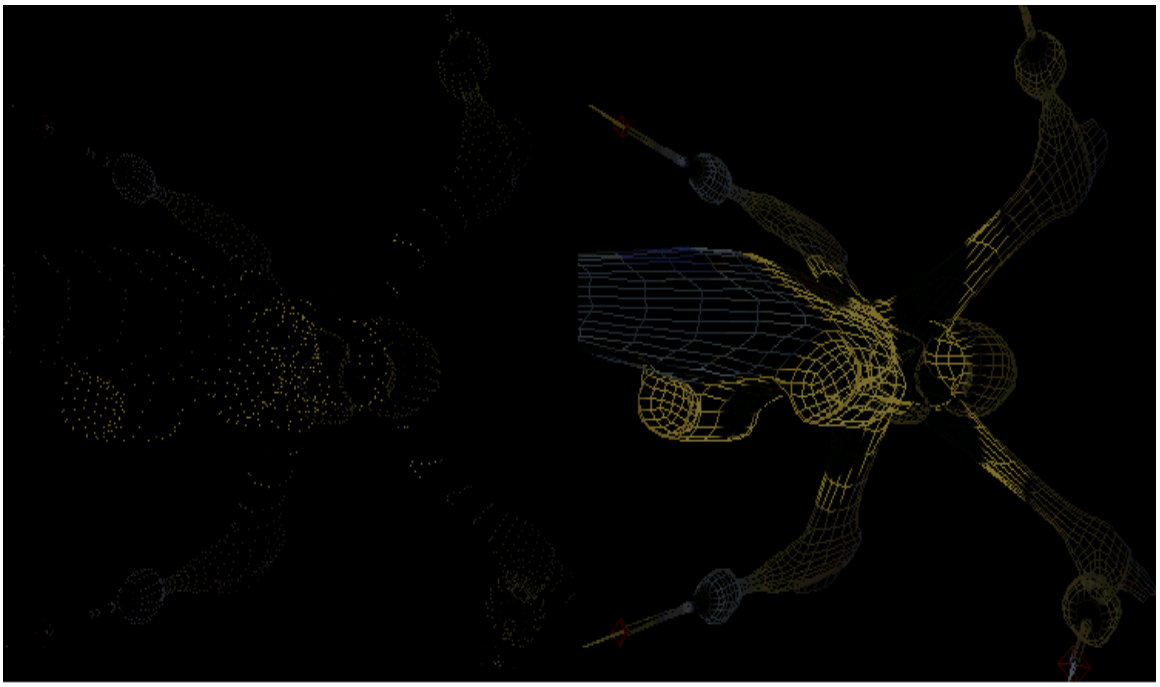

Plate 4.1 – A complex 3-D computer graphics model. On the left hand side only the 3-D vertices are shown. On the right hand is shown how the model is made up of polygons.

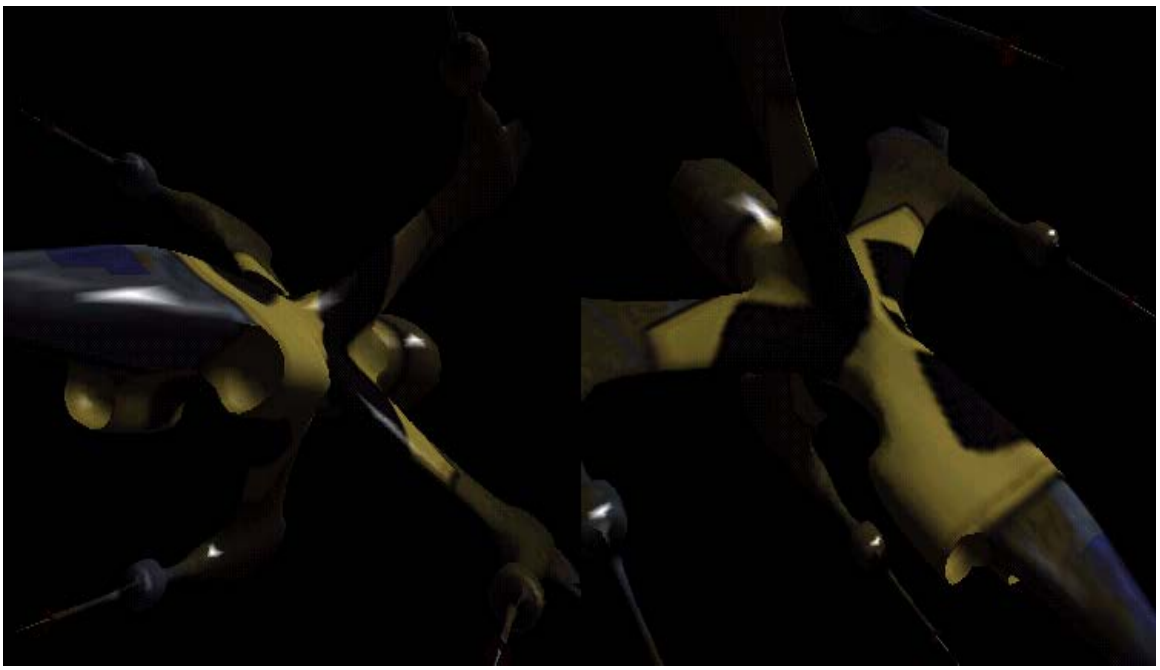

Plate 4.2 – A complex 3-D computer graphics model.

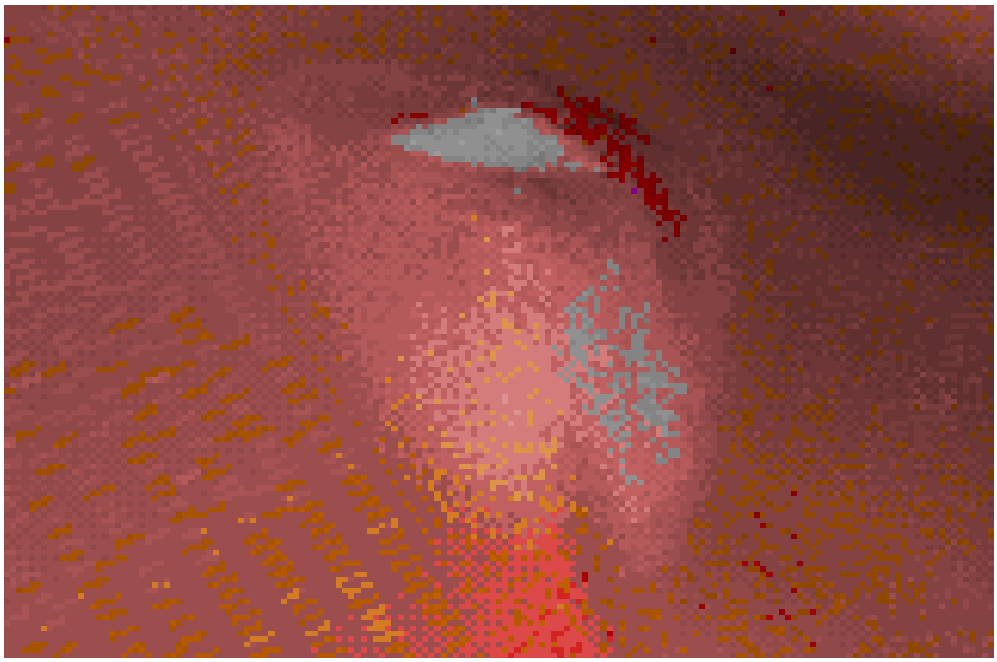

Plate 4.3 – Ulcer rendered with 256 colour palette.

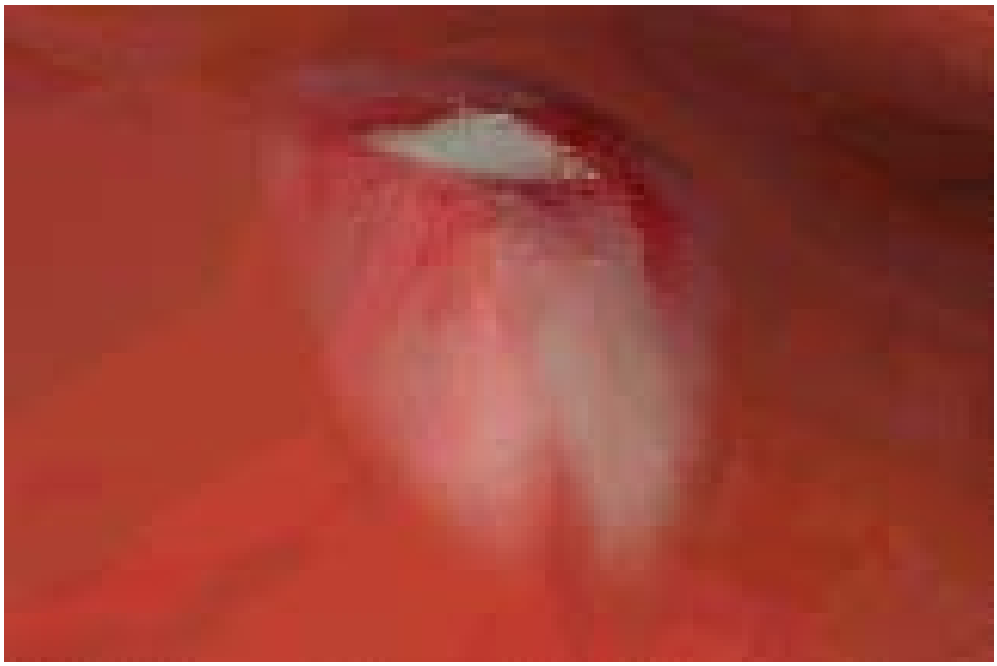

Plate 4.4 – Ulcer rendered with 16-bit high colour.

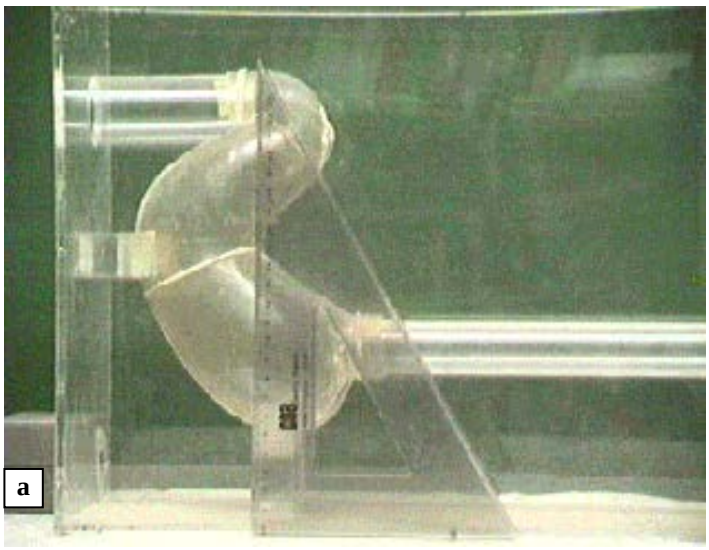

Plate 4.5 – To register the computer model, measurements of the physical model must be taken. The symbol "a" indicates the 3-D tracking device's transmitter.

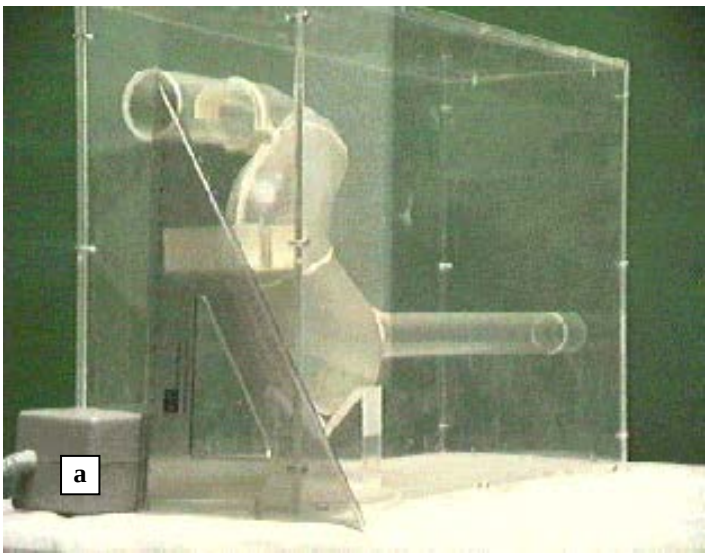

Plate 4.6 – To register the computer model, measurements of the physical model must be taken. The symbol "a" indicates the 3-D tracking device's transmitter.

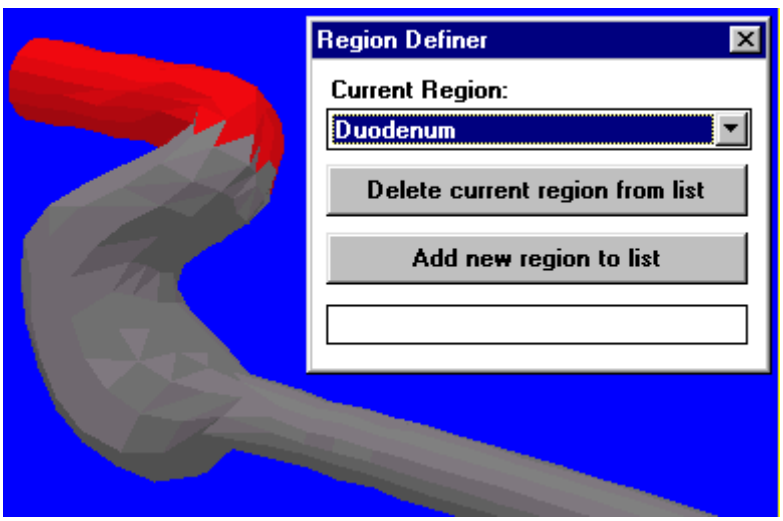

Plate 5.1 – Example of the region definer application. The polygons shown in red were selected by the user to be part of the duodenum.

Plate 6.1 – A description of the above image in an endoscopic atlas would be: “These polypoid lesions occur multiply or singly and have smooth overlying mucosa resembling that of normal a stomach. Typically discovered as incidental findings at endoscopy, these are not true polyps, rarely cause symptoms in the absence of inflammation, and have no malignant potential. They cannot, however, be reliably distinguished from adenomastous polyps on the basis of gross appearance.”

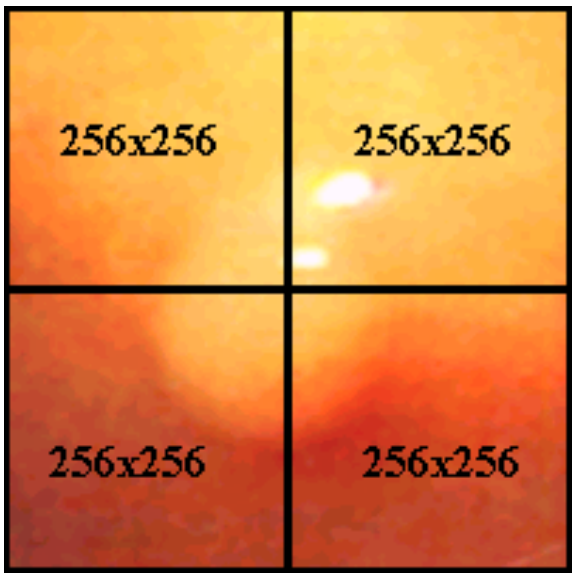

Plate 6.2 – An image of an abnormal condition divided into squares of 256 by 256.

Plate 6.3 – Computer image of a real ulcer.

Plate 6.4 – The image of a real ulcer has been cut and placed onto an image of normal stomach tissue that can be tiled. Note the definite circle.

Plate 6.5 – The result after the circle line of the ulcer has been blended onto the background image of the normal stomach tissue.

Plate 6.6 – An unprocessed ulcer that does not blend onto the background.

Plate 6.7 – A processed ulcer that blends onto the background tissue.

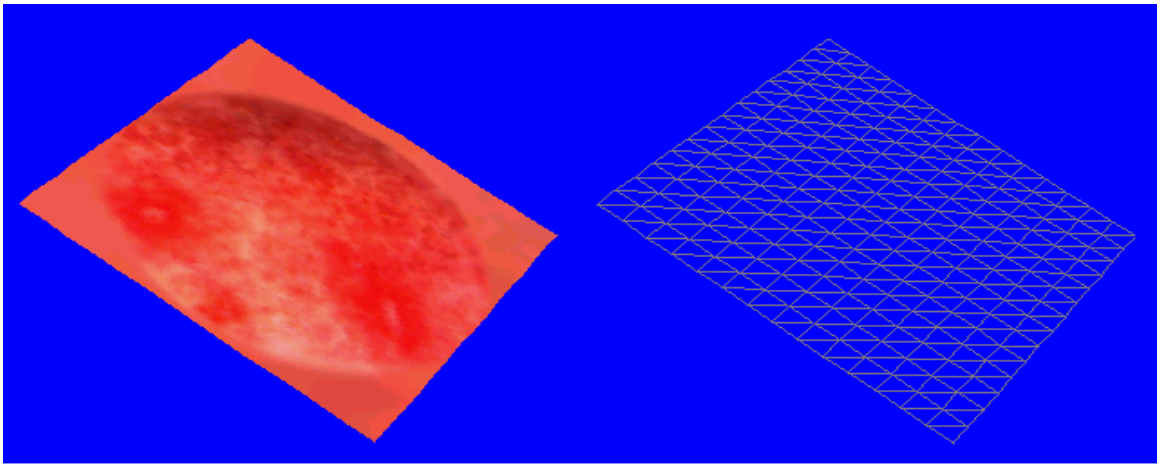

Plate 6.8 – Flat 3-D grid with computer image of Gastritis.

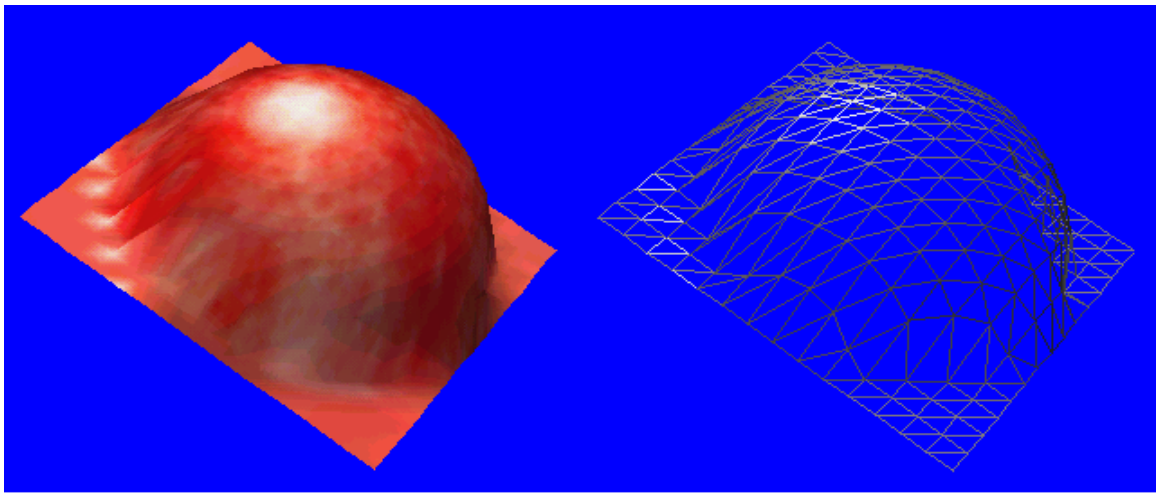

Plate 6.9 – 3-D grid altered with a mathematical function.

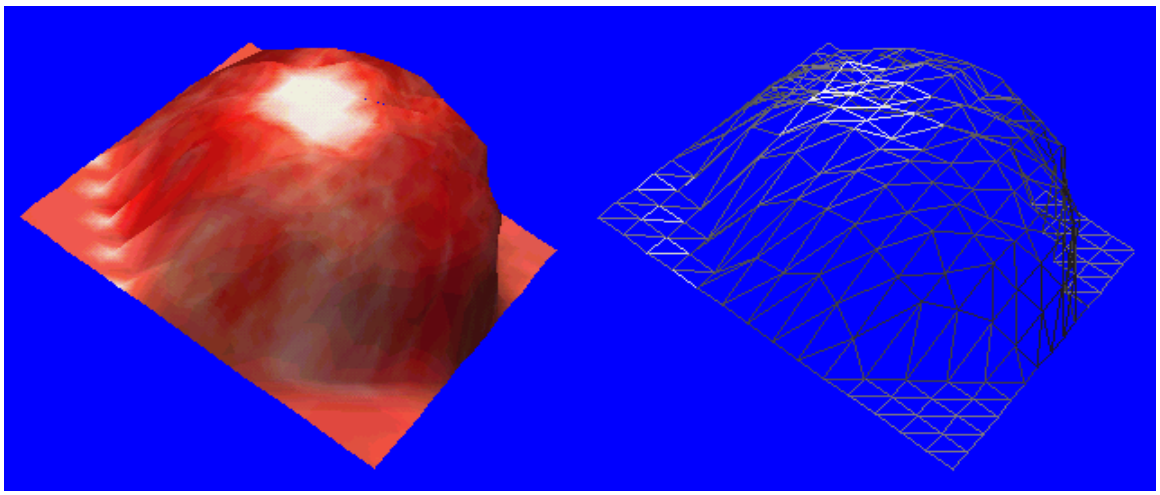

Plate 6.10 – 3-D grid with a mathematical function and randomly distorted.

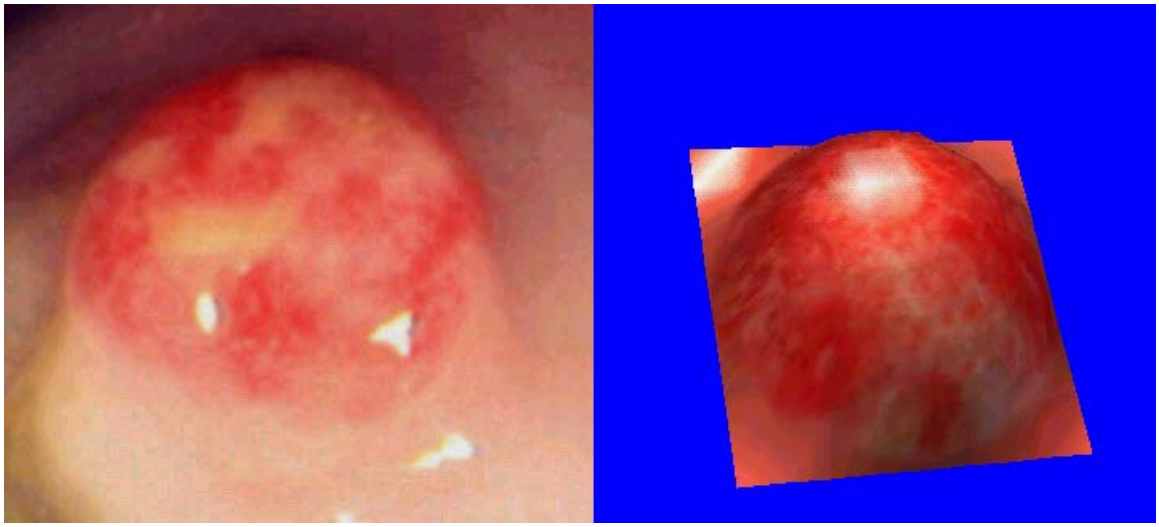

Plate 6.11 – On the left side is a photo taken of a carcinoid (benign tumor) and on the right side is a computer generated 3-D condition.

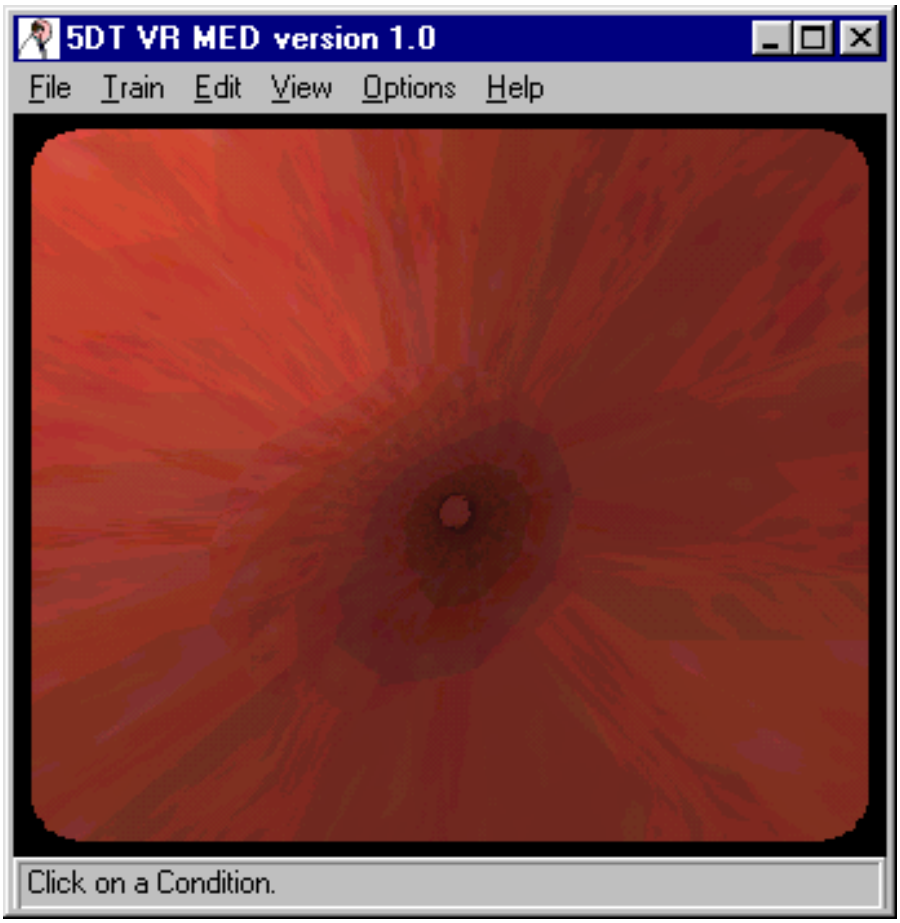

Plate 7.1 – The computer graphic user interface. On top is the main menu. At the bottom is an area for messages to the user and in between is the image of the VR model.

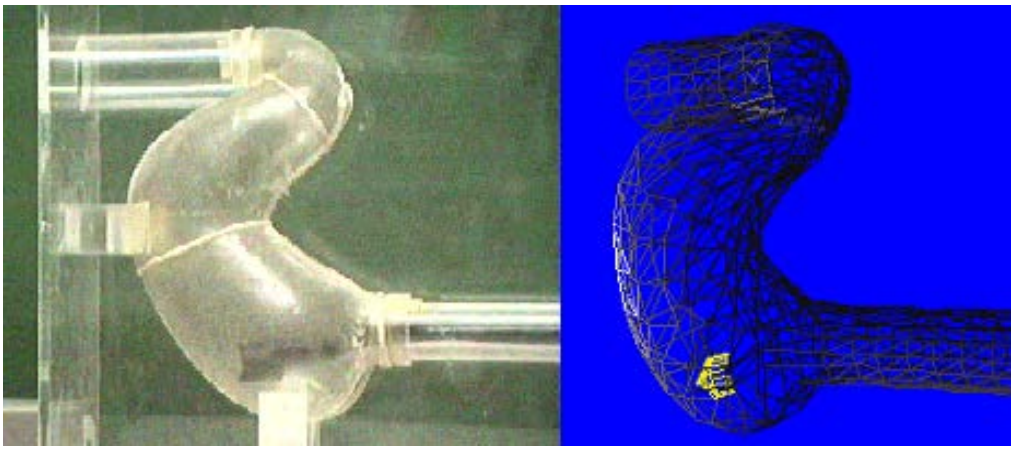

Plate 7.2 – On the left side is a photo of the physical model and on the right hand is the camera orientation view window.

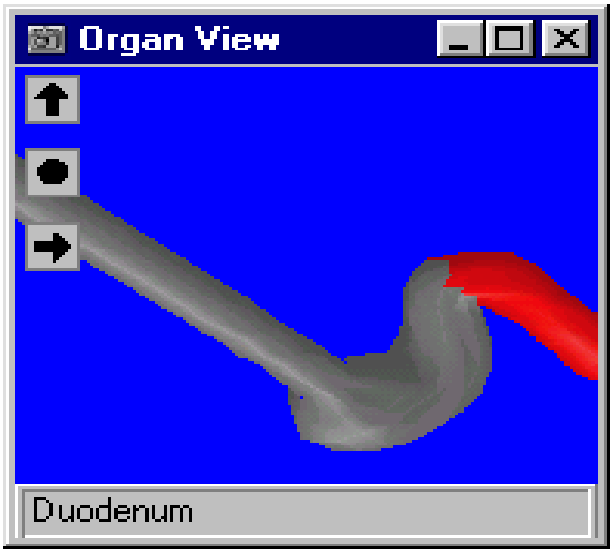

Plate 7.3 – Example of the organ view window.

Plate 7.4 – Example of the video recorder window.

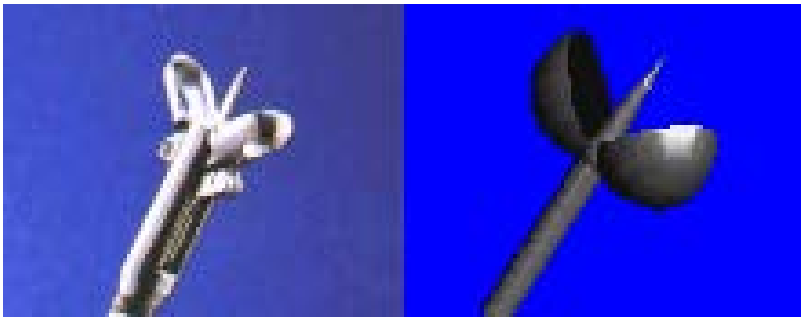

Plate 7.5 – On the left side is a photo of a real biopsy tool, on the right hand side is a computer generated biopsy tool.

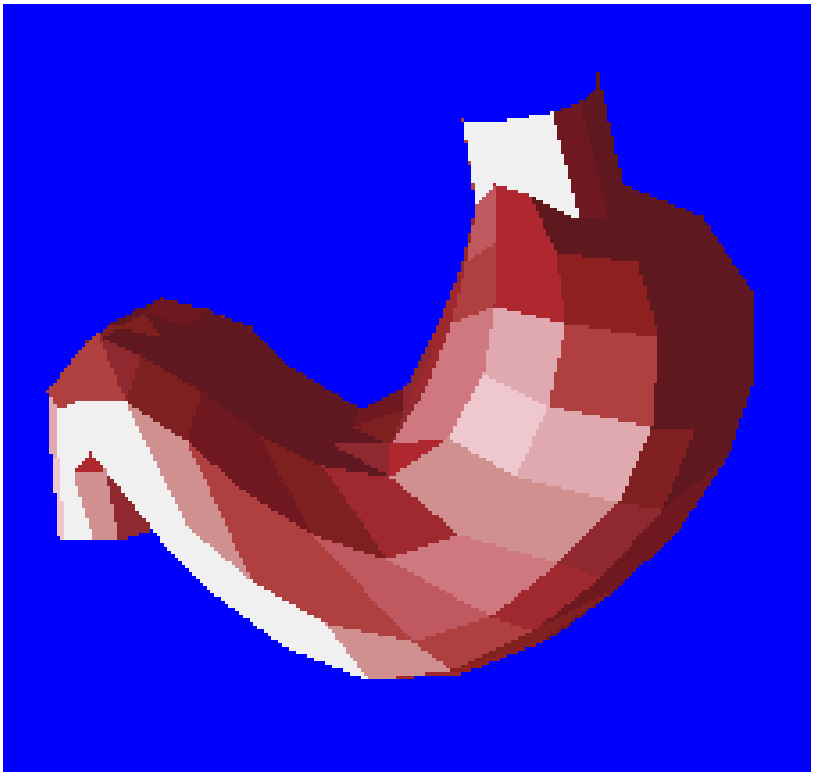

Plate 7.6 – Example of flat shading.

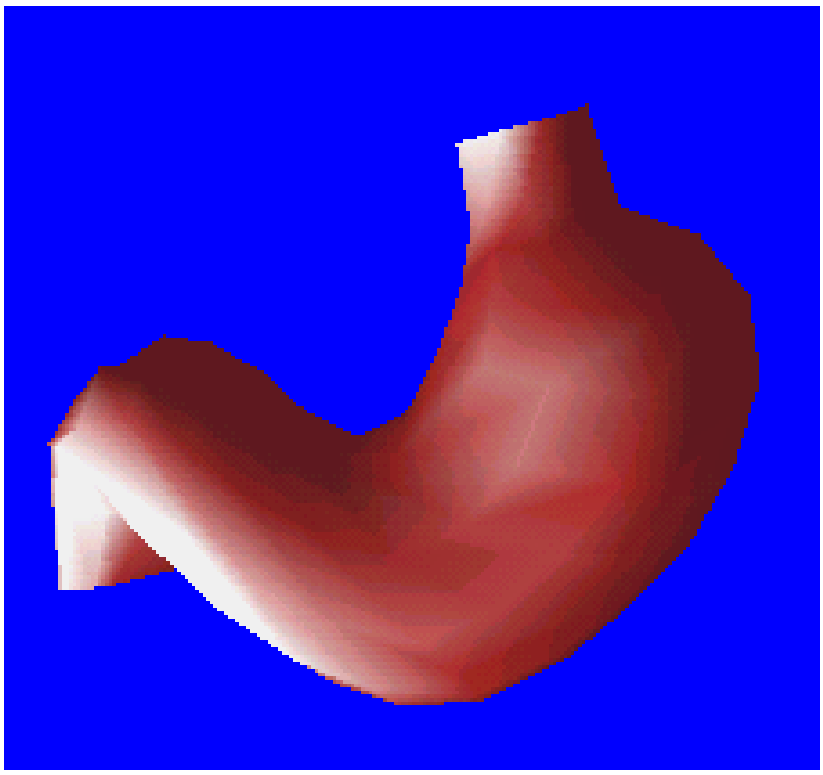

Plate 7.7 – Example of smooth shading.

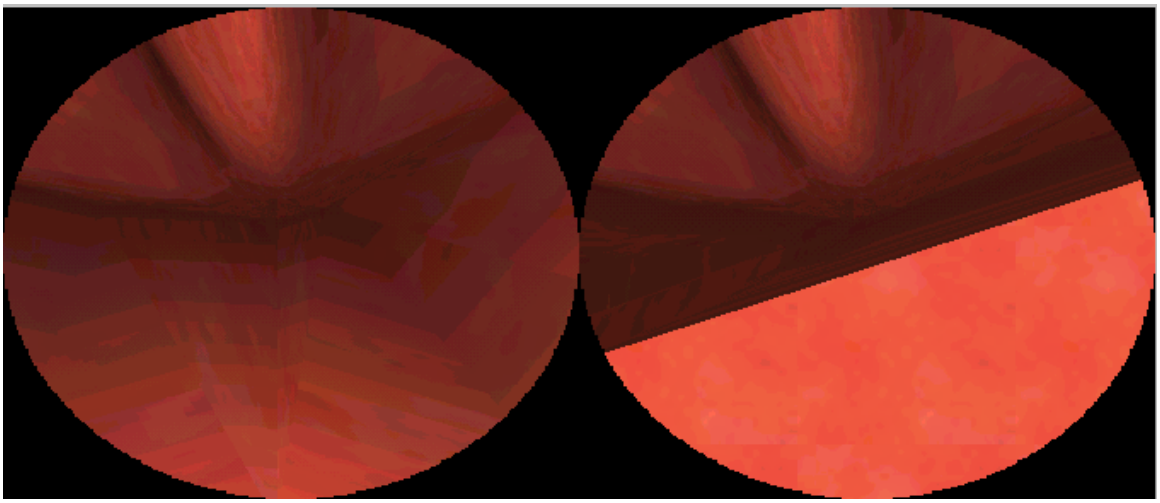

Plate 7.8 – On the left side is the system with warping and on the right hand side is the system without warping. On the right hand, the light background colour can be seen where the tip of the gastroscope penetrates the VR model.

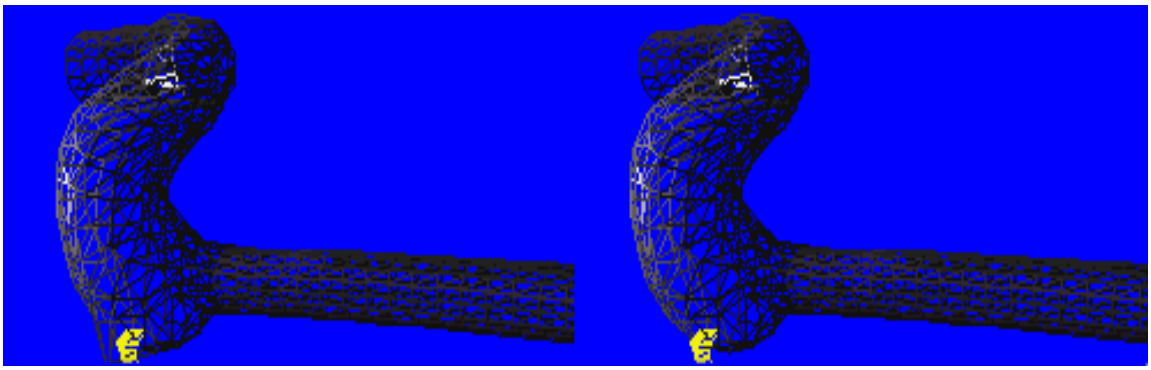

Plate 7.9 – On the left side is the system with warping and on the right hand side is the system without warping. On the left hand can be seen how some vertices are warped away so that the tip of the gastroscope cannot penetrate the VR model.

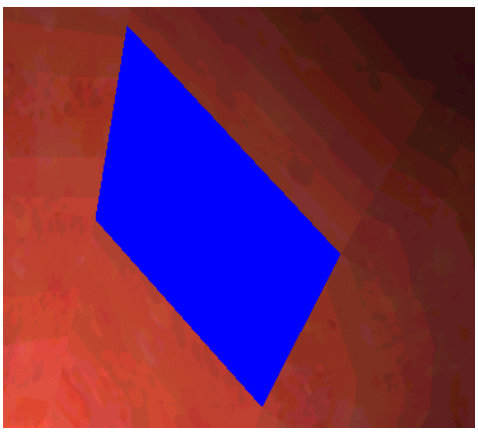

Plate 7.10 – The hole in the VR model after a polygon was deleted.

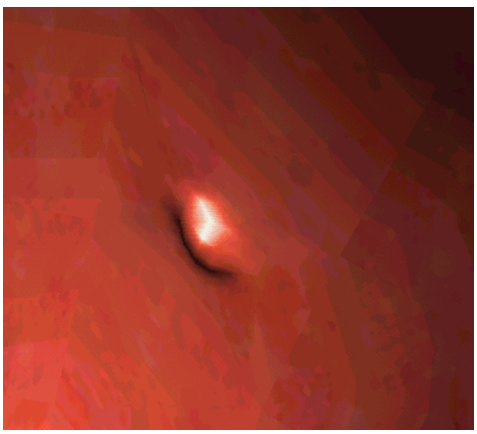

Plate 7.11 – A 3-D condition is fitted into the hole where a polygon was deleted.

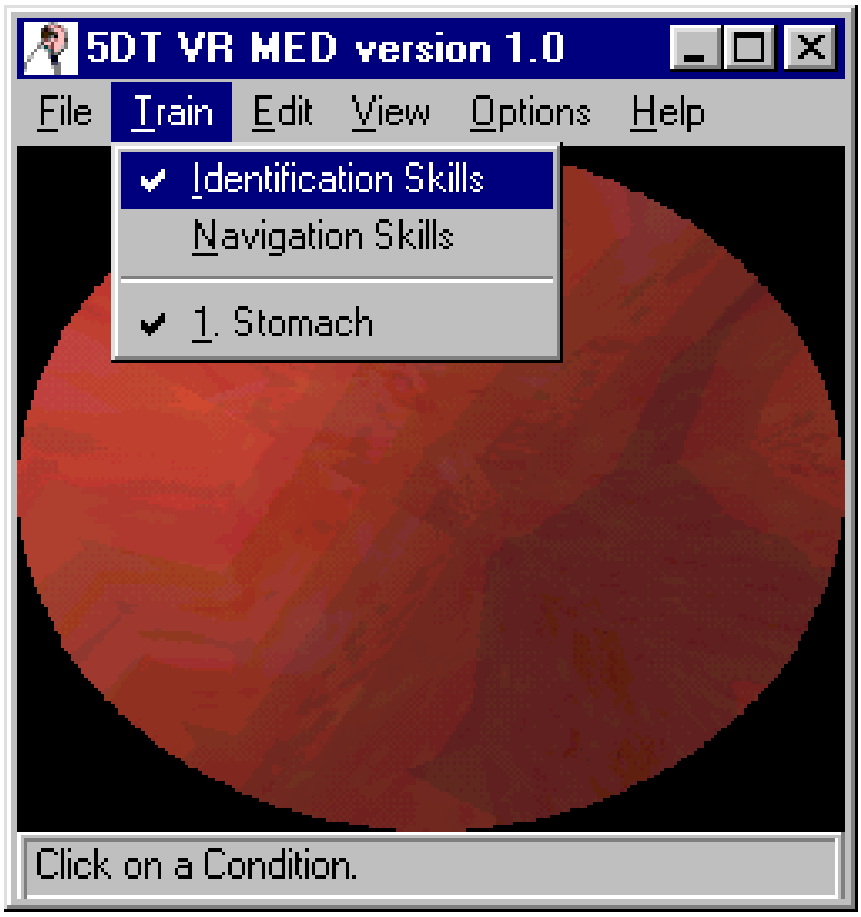

Plate 8.1 – Checked menu item for identification training.

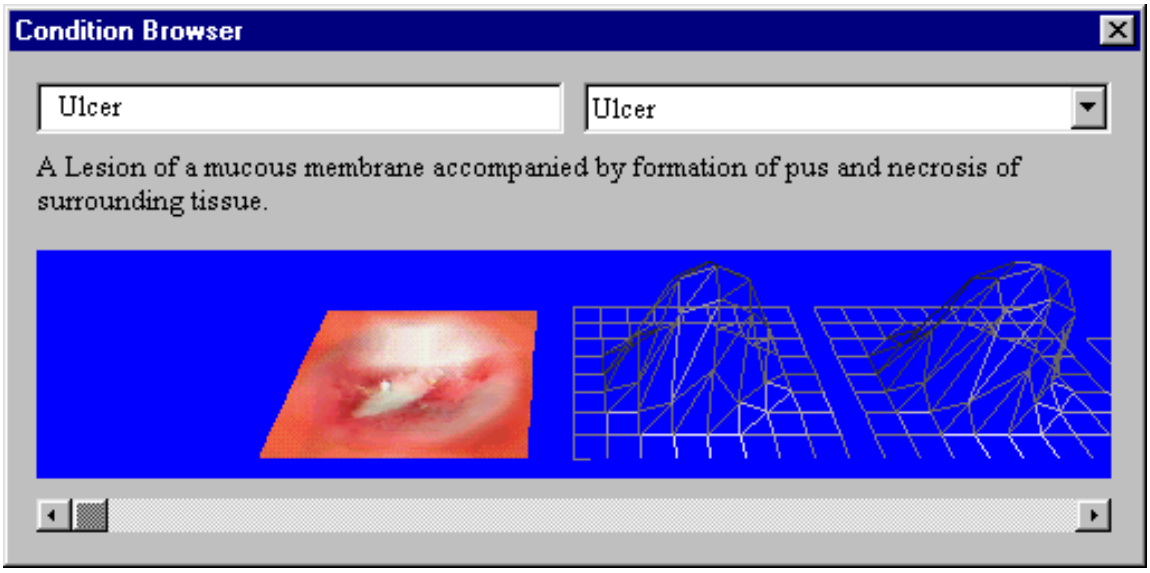

Plate 8.2 – 3-D Condition browser showing the ulcer as the currently selected condition.

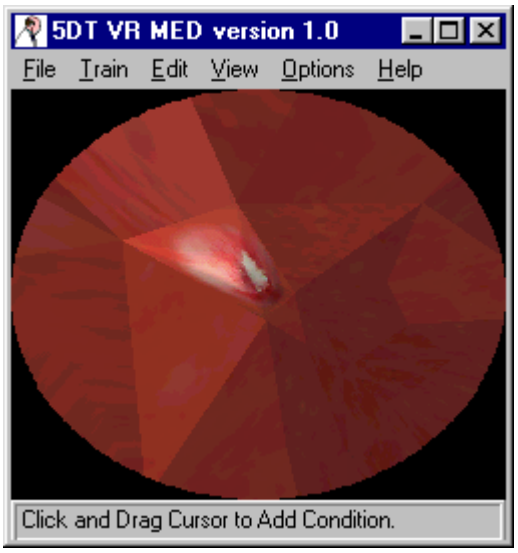

Plate 8.3 – Ulcer placed inside stomach. Note the blocky appearance when the system is in edit mode.

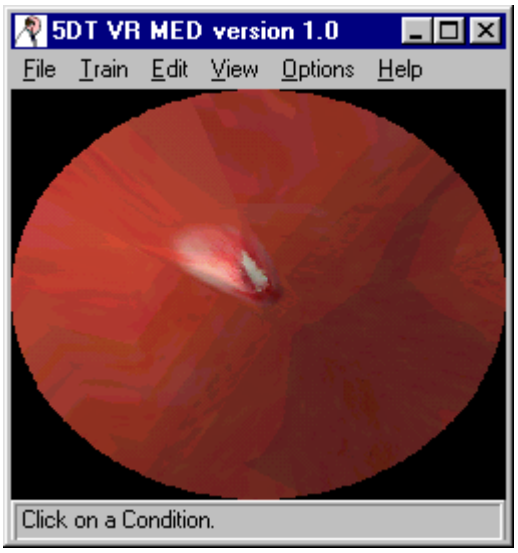

Plate 8.4 – Ulcer placed inside the stomach. Note the smooth appearance when the system is not in edit mode.

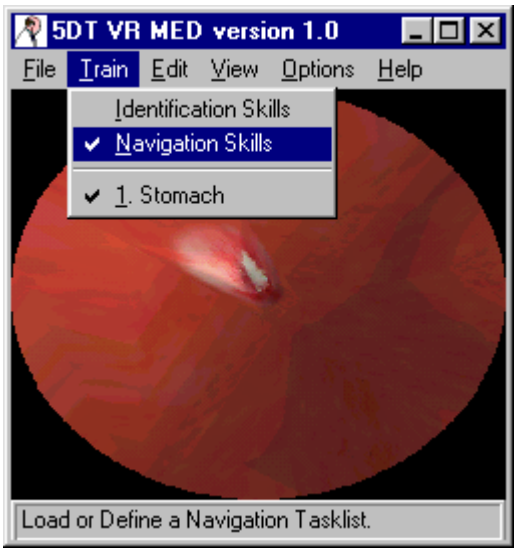

Plate 8.5 – Checked menu item for navigation training.

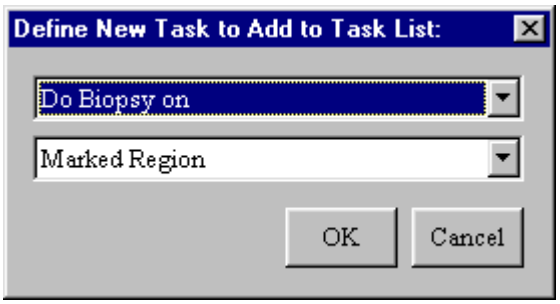

Plate 8.6 – Navigation task for biopsy taking.

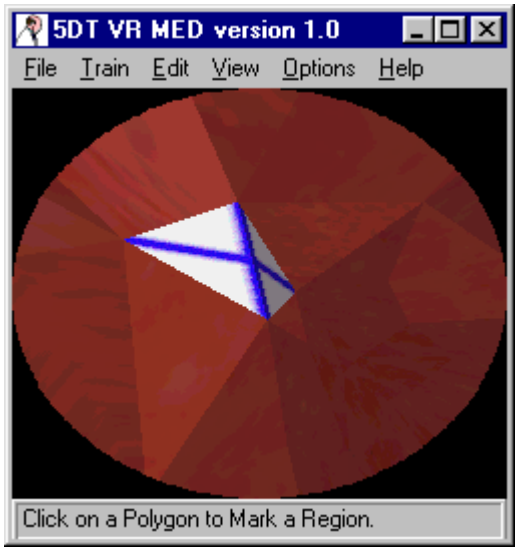

Plate 8.7 – Marked region for biopsy taking.

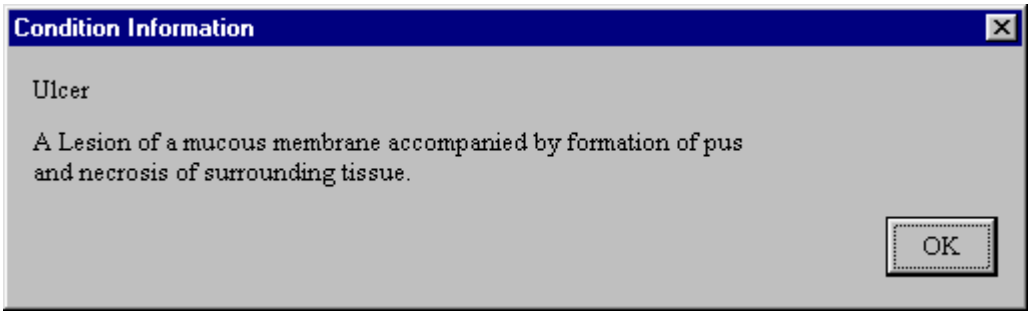

Plate 8.8 – When in learn mode, the trainee will be given information about conditions.

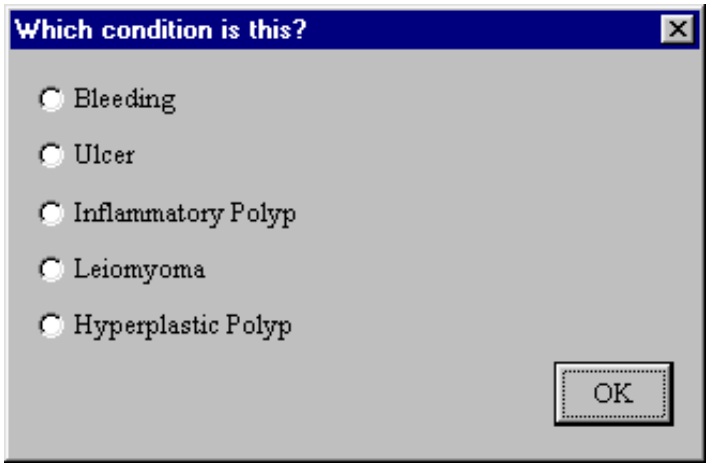

Plate 8.9 – When in test mode the trainee will be asked multiple-choice questions.

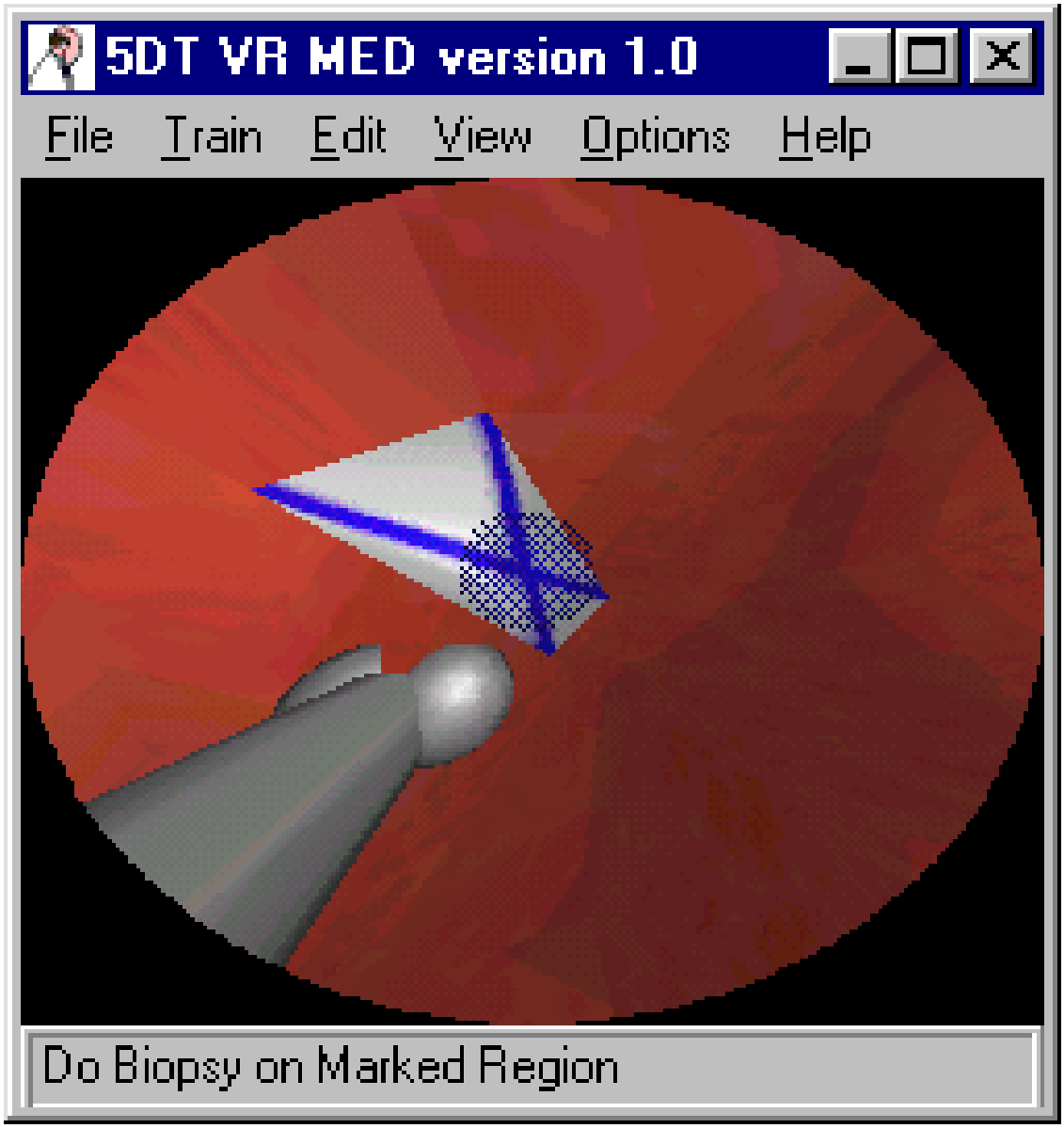

Plate 8.10 – Main window showing a message at the bottom about the navigation task to be completed. A biopsy tool reaching toward a marked region can also be seen.

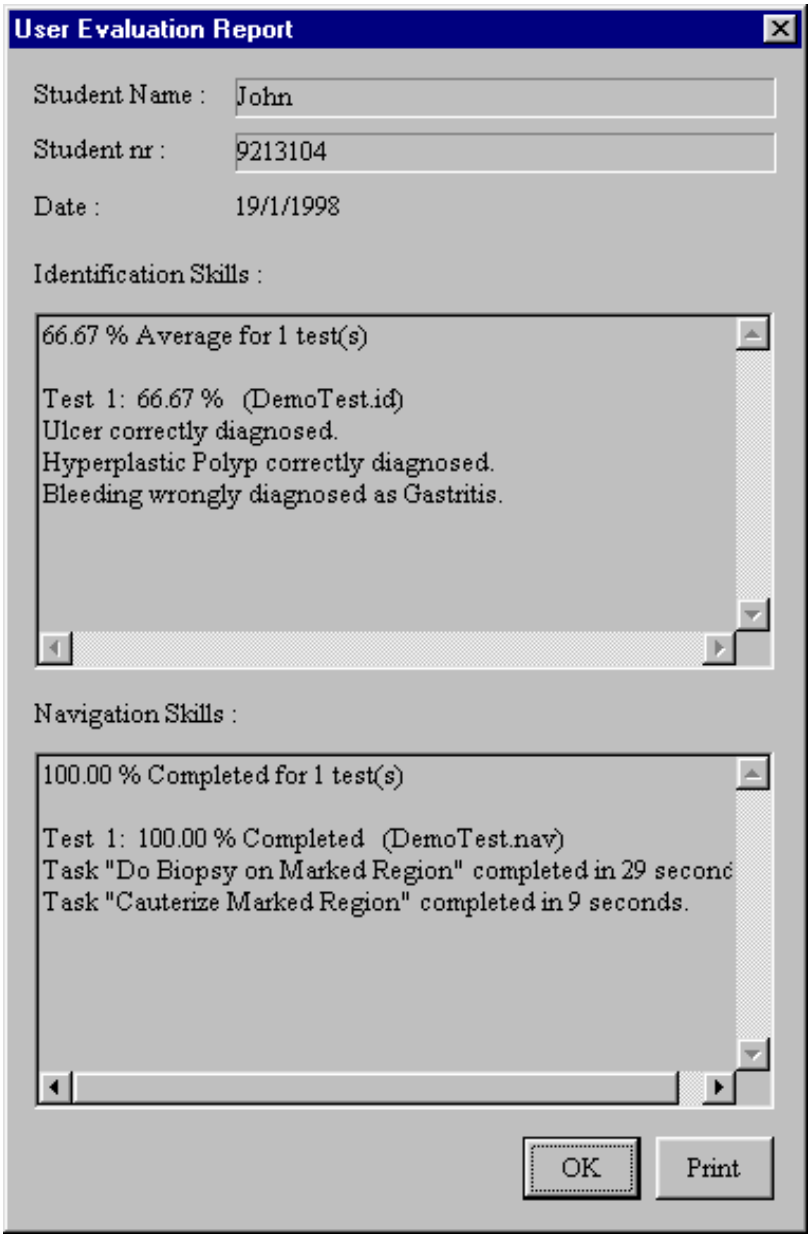

Plate 8.11 – Evaluation report.