FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

UNIVERSITY OF MALAYA

KUALA LUMPUR

Prepared By :

Student : Kee Meng Yit
Course : Bachelor of Computer Science
Matrix No : WEK 010110
Title : Virtual Reality Training in Laparoscopic Surgery

Approved By :

Name : Dr. Rosli Salleh
Position : Project Supervisor
Signature :

Approved By :

Name : Ms. Rafidah Md. Noor
Position : Project Moderator
Signature :

Abstract

The second millennium has brought with it a new era of modern surgery. The creation of video surgery is as revolutionary to this century as the development of anesthesia and sterile technique was to the last one. With ten years of solid experience behind them, surgeons can now confidently approach almost every part of the human body with cameras and video monitors.

Today more than 90% of all gallbladder surgery is performed laparoscopically, leaving patients with only four tiny marks and minimal discomfort. These dramatic patient benefits make laparoscopic gallbladder surgery the procedure of choice among both surgeons and patients.

It is increasingly important for surgeons to practice techniques outside the operating room. These techniques include basic procedural training, allowing for as much repetition as needed, as well as rare critical events. Critical events can be so rare that occasional practice of them is necessary.

Currently, a variety of patient simulators allow students to experience life-like medical scenarios in a controlled environment. They provide a medium for instruction in physiology, technical skills, analytical clinical reasoning, and group crisis management.

Laparoscopic cholecystectomy (gallbladder surgery) constitutes the basis of all the educational programmes on laparoscopic techniques. This operation and all the other laparoscopic operations begin with the same basic steps.

This project will attempt to provide a reasonably realistic virtual reality surgical training environment for a laparoscopic cholecystectomy surgery.

Acknowledgements

Firstly, Bravo! To Faculty of Computer Science and Information Technology, University of Malaya, for having such a wonderful and meaningful program that would allow students to gain theoretical and practical knowledge in a 'real-time' project environment.

I would like to express my utmost gratitude to Dr. Rosli Salleh for supervising me and thus has given me a world of opportunities to learn and excel in many ways. To Cik Rafidah binti Md. Noor, my project's moderator, I would like to thank her for the valuable advice and ideas that she has given me.

I would always be indebted to both of them for all the knowledge, ideas, support, assistance and guidance that they had given me for the whole duration of this project. It has been truly unforgettable and wonderful experience learning from the both of you.

Table of Contents

List of Tables and Figures

List of Appendices

CHAPTER 1 INTRODUCTION

1.1 OVERVIEW

1.2 PROJECT'S DEFINITION

1.3 PROJECT'S OBJECTIVES

1.4 PROJECT'S SCOPES

1.5 PROJECT'S SCHEDULE

1.0 Introduction

1.1 Overview

The second millennium has brought with it a new era of modern surgery. The creation of video surgery is as revolutionary to this century as the development of anesthesia and sterile technique was to the last one. With ten years of solid experience behind them, surgeons can now confidently approach almost every part of the human body with cameras and video monitors. In "Laparoscopic Surgery," Medina (2003) explained:

> First they make a small cut in the skin and then introduce a harmless gas, such as carbon dioxide, into the body cavity to expand it and create a large working space. Through additional small cuts, a rod shaped telescope, attached to a camera, and other long and narrow surgical instruments are place into the newly formed space. By this means, under high magnification diseased organs are able to be examined with minimal trauma to the patient.

It is increasingly important for surgeons to practice techniques outside the operating room. These techniques include basic procedural training, allowing for as much repetition as needed, as well as rare critical events. Critical events can be so rare that occasional practice of them is necessary.

Currently, a variety of patient simulators allow students to experience life-like medical scenarios in a controlled environment. They provide a medium for instruction in physiology, technical skills, analytical clinical reasoning, and group crisis management.

By using the latest in PC graphics technology, the visualization of the laparoscopic surgery simulator can provide realism that was until recently only possible in high end, expensive computers (Upton, 2003).

1.2 Project's Definition

1.2.1 The Problem

In "Virtual Reality Based Surgery Simulation for Endoscopic Gynaecology," Szekely,G. et al. (2003) discusses the advantages and disadvantages of a gynecologic laparoscopic surgery:

> The relatively large cuts in open surgery can be replaced by small perforation holes, serving as entry points for optical and surgical instruments. The small spatial extent of the tissue injury and the careful selection of the entry points result in a major gain in patient recovery after operation.
>
> The price for these advantages is paid by the surgeon who loses direct contact with the operation site. The necessary visual information is mediated by a (usually monoscopic) specialized camera (the laparoscope) and is presented on a screen, distracting normal hand-eye coordination. Due to geometrical constraints posed by the external control of the surgical instruments through the trocar hull, the surgeon loses much of the manipulative freedom usually available in open surgery.

Performing operations under these conditions demands very specific capabilities of the surgeon, which can only be gained with extensive training. Virtual reality based surgical simulator systems offer a very elegant solution to this training problem.

1.2.2 The Solution

Laparoscopic cholecystectomy (gallbladder surgery) is the most frequently laparoscopic operation performed and constitutes the basis of all the educational programmes on laparoscopic techniques. This operation and all the other laparoscopic operations begin with the same basic steps.

This project will attempt to provide a reasonably realistic virtual reality surgical training environment for a laparoscopic cholecystectomy surgery.

1.3 Project's Objectives

Core objectives of the project are as below.

- To create a Virtual Reality training tool for a laparoscopic cholecystectomy surgery using OpenGL
- To provide a safe, controllable environment for users to learn, allowing them to make mistakes without consequences to the patient
- To explore the extendibility and reliability of OpenGL as a high level graphics programming language.

1.4 Project's Scopes

The project's scopes are as below.

• Simulation of a laparoscopic cholecystectomy surgery that includes procedures from the basic instrument control to effects such as liquid pooling.

• The training tool would be a standalone program.

• Due to the expensive and unavailability of hardware such as robotic arms, simulation controls will be constraint to only keyboard and mouse.

1.5 Project's Schedule

This project was divided into two phases. The duration of Phase 1 was from June 2003 till the end of August 2003 while Phase 2 began at September 2003 and ended at February 2003. Figure 1.1 below depicts the Gantt Chart for this project.

Phase 1 comprised of Preliminary Study and Planning, Literature Review, System Analysis and System Design.

Meanwhile Prototyping, Development/Coding, Testing and Review, Implementation and Maintenance were done during Phase 2.

Documentation was done continuously throughout both phases.

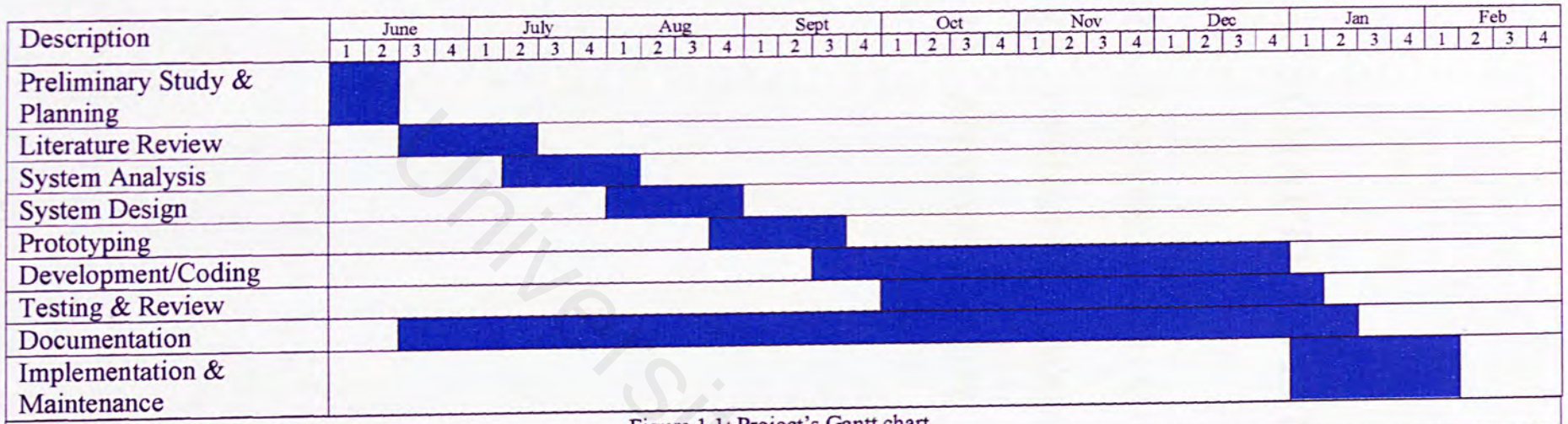

Figure 1.1: Project's Gantt chart

CHAPTER 2 LITERATURE REVIEW

2.1 LAPAROSCOPIC SURGERY

2.2 OPENGL PROGRAMMING

2.3 ANALYSIS STUDY

2.0 Literature Review

2.1 Laparoscopic Surgery

2.1.1 Overview

2.1.1.1 What Is Laparoscopy?

According to Your Medical Resource (2003):

> Laparoscopy (pronounced "lap-a-ROSS-coe-pee") is a surgical procedure performed through very small incisions in the abdomen, using specialized instruments. A pencil-thin instrument called a laparoscope is used, and it gives the surgeon an exceptionally clear view, on a TV monitor, of the inside of the abdominal cavity.
>
> A laparoscope has lenses like a telescope to magnify body structures, a powerful light to illuminate them, and a miniature video camera. The camera sends images of the inside of the body to a TV monitor in the operating room. Specialized surgical instruments can be inserted through the laparoscope, and through small incisions nearby.
>
> This type of surgery is called 'minimally invasive' because of the very small incisions used. Yet major procedures can now be performed using this technique. The term laparoscopy is used when this type of surgery is performed in the abdomen. It's called arthroscopy when performed in a joint, and endoscopy when done through a natural opening in the body, such as the mouth or nose.

- "Laparo" comes from a Greek word meaning "flank," which is the side of the body between the ribs and hips. Doctors use this term to refer to the abdomen. The term "scope" means to look at or examine.
- Many procedures once done through a large opening in the abdomen can now be done with the small incisions of laparoscopy.
- Laparoscopy has become the preferred surgical technique for some conditions, such as gallbladder disease.

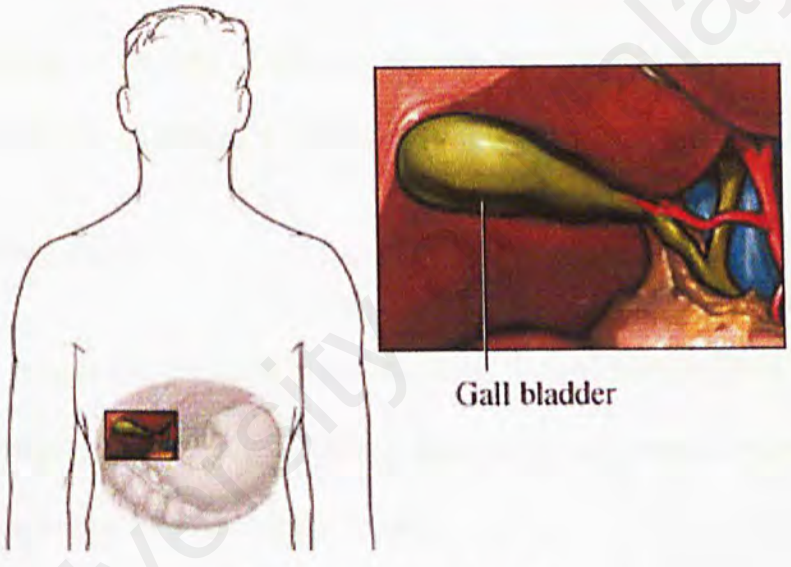

Figure 2.1: The Gall bladder

2.1.1.2 Advantages of Laparoscopic Surgery

Laparoscopy is easier on the patient because it uses a few very small incisions. For example, traditional "open surgery" on the abdomen usually requires a four- to five-inch incision through layers of skin and muscle. In laparoscopic surgery, the doctor usually makes two to three incisions that are about a half-inch long.

The smaller incisions cause less damage to body tissue, organs, and muscles so that the patient

- can go home sooner

  Depending on the kind of surgery, patients may be able to return home a few hours after the operation, or after a brief stay in the hospital.

- recovers quickly

  Many people can return to work and their normal routine three to five days after surgery. In contrast, traditional laparotomy may require a person to limit daily activities for four to eight weeks.

- experiences fewer post-operative complications and less pain

  The amount of discomfort varies with the kind of surgery. In most cases, however, patients feel little soreness from the incisions, which heal within a few days. Most need little or no pain medicine.

- has less scaring

  The incisions for most kinds of laparoscopic surgery heal without noticeable scars. In laparoscopic surgery on a woman's reproductive system, for

instance, one incision usually can be hidden in the belly button area. The others can be placed low in the abdomen, where any scars would be covered by a bikini.

2.1.1.3 Possible Risks of Laparoscopic Surgery

Since laparoscopy involves minimal damage to body tissues, it is generally safer than open operations. In diagnostic laparoscopy, for instance, complications occur in about three out of every 1,000 operations, a significantly lower number than traditional surgery (Your Medical Resource, 2003). A complication is an unforeseen problem that occurs during or after surgery, such as internal bleeding or injury to a healthy organ.

Possible complications of laparoscopy:

- Risks for any type of surgery may be greater for people who are obese, smoke cigarettes, or have additional health problems.
- Laparoscopy usually requires general anesthesia, which carries certain risks. Modern general anesthesia, however, is safe and reactions are rare. The individual must be sure to tell the doctor if he or she had a bad reaction to anesthesia in the past, or if a close family member has experienced such a reaction.
- Injury to blood vessels or organs, which causes bleeding.
- Damage to ducts or other structures that allow body fluids to leak out.

Sometimes the surgery cannot be successfully completed by laparoscopy. Then the doctor may have to complete the operation using traditional "open" abdominal surgery, called laparotomy. This is called "converting" to laparotomy (Your Medical Resource, 2003).

2.1.2 Types of Laparoscopic Surgery

There are 3 types of laparoscopic surgery.

- **Diagnostic laparoscopy** is used to determine the cause of an abdominal problem or sometimes to provide additional information after other tests have been performed.
- **Pelvic** or **gynecologic laparoscopy** is used both for diagnostic purposes when there is pain or infection, and for surgery such as tubal ligation (having the fallopian tubes "tied") or removal of ovarian cysts or other abnormal pelvic growths.
- **Intra-abdominal laparoscopic** surgery may be used for appendectomies, gallbladder surgery, hernia repair and other procedures.

2.1.3 Laparoscopic Surgery Procedures

The procedures involved for laparoscopic surgery are both methodical and detailed. This section outlines each of the intended simulation procedures that will replicate the actual surgery from the basic instrument control to effects such as liquid pooling.

These procedures form part of a detailed requirements analysis in Chapter 4. Not all of these procedures will be incorporated into the virtual reality training simulator.

In "Laparoscopic Surgery Simulation Realism In A PC," Upton (2003) outlines and discusses these procedures in detail:

- Instrument Insertion and Removal
- Tissue Deformation, Translation, Rotation
- Tissue Cutting, Tearing, and Puncture
- Electro-Cautery
- Contact Laser
- Non-Contact Laser
- Smoke Accumulation
- Surgical Stapler
- Surgical Clip Application
- Suction
- Irrigation
- Bleeding
- High-Pressure Bleeding
- Low-Pressure Bleeding
- Liquid Pooling

2.2 OpenGL Programming

2.2.1 Overview

2.2.1.1 What Is OpenGL? (Woo et al. ,1997)

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that can be use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might be allowed to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, desired model are build from a small set of geometric primitives - points, lines, and polygons.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

The following list briefly describes the major graphics operations which OpenGL performs to render an image on the screen.

1. Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.)
2. Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene.
3. Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pasting a texture onto the objects, or some combination of these three actions.
4. Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called *rasterization.*

During these stages, OpenGL might perform other operations, such as eliminating parts of objects that are hidden by other objects. In addition, after the scene is rasterized but before it's drawn on the screen, you can perform some operations on the pixel data if you want.

In some implementations (such as with the X Window System), OpenGL is designed to work even if the computer that displays the graphics created isn't the computer that runs the graphics program. This might be the case if one is working in a networked computer environment where many computers are connected to one another by a digital network. In this situation, the computer on which the program runs and issues OpenGL drawing commands is called the client, and the computer that receives those

commands and performs the drawing is called the server. The format for transmitting OpenGL commands (called the *protocol*) from the client to the server is always the same, so OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server.

2.2.1.2 OpenGL as a State Machine (Woo et al. ,1997)

OpenGL is a state machine. They are put into various states (or modes) then remain in effect until they are changed. The current color is a state variable. The current color can be set to white, red, or any other color, and thereafter every object is drawn with that color until the current color is set to something else. The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations, line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**.

Each state variable or mode has a default value, and at any point the system can be queried for each variable's current value. Typically, one of the six following commands can be used to do this: **glGetBooleanv()**, **glGetDoublev()**, **glGetFloatv()**, **glGetIntegerv()**, **glGetPointerv()**, or **glIsEnabled()**.Some state variables have a more specific query command (such as **glGetLight*()**, **glGetError()**, or **glGetPolygonStipple()**). In addition, a collection of state variables can be saved on an attribute stack with **glPushAttrib()** or **glPushClientAttrib()**, temporarily modify them, and later restore the values with **glPopAttrib()** or **glPopClientAttrib()**. For temporary state changes, these commands should be used rather than any of the query commands, since they're likely to be more efficient.

See Appendix A for the complete list of state variables that can be queried. For each variable, the appendix also lists a suggested **glGet*()** command that returns the

variable's value, the attribute class to which it belongs, and the variable's default value.

2.2.1.3 OpenGL Rendering Pipeline (Woo et al. ,1997)

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 2.2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do.

The following diagram shows the Henry Ford assembly line approach, which OpenGL takes to processing data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the framebuffer.

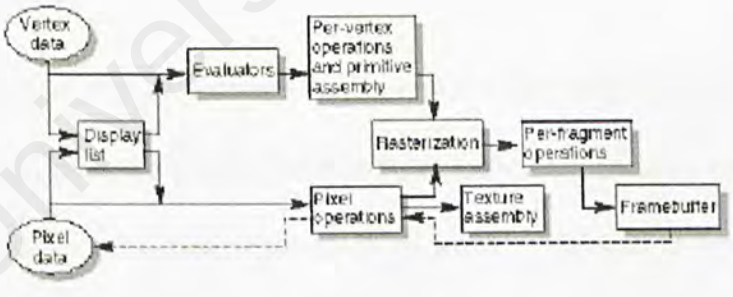

Figure 2.2: OpenGL -Order of Operations

Display Lists

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (The alternative to retaining data in a display list is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on the screen. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects.

Then viewport and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines.

The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data

is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the framebuffer.

Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them.

Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

Rasterization

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

Fragment Operations

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled. The first operation which may be encountered is texturing, where a texel (texture element) is generated from texture memory for each fragment and applied to the fragment. Then fog calculations may be applied, followed by the scissor test, the alpha test, the stencil test, and the depth-buffer test (the depth buffer is for hidden-surface removal). Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processed fragment is drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

2.2.2 OpenGL Utility Toolkit (GLUT)

2.2.2.1 What is GLUT? (Kilgard, 1996)

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C and FORTRAN bindings for writing window system independent OpenGL programs. The toolkit supports the following functionality:

- Multiple windows for OpenGL rendering.
- Callback driven event processing.
- Sophisticated input devices.
- An "idle" routine and timers.
- A simple, cascading pop-up menu facility.
- Utility routines to generate various solid and wire frame objects.
- Support for bitmap and stroke fonts.
- Miscellaneous window management functions, including managing overlays.

2.2.2.2 Background (Kilgard, 1996)

One of the major accomplishments in the specification of OpenGL was the isolation of window system dependencies from OpenGL's rendering model. The result is that OpenGL is window system independent.

Window system operations such as the creation of a rendering window and the handling of window system events are left to the native window system to define. Necessary interactions between OpenGL and the window system such as creating and binding an OpenGL context to a window are described separately from the OpenGL specification in a window system dependent specification. For example, the GLX specification describes the standard by which OpenGL interacts with the X Window System.

The predecessor to OpenGL is IRIS GL. Unlike OpenGL, IRIS GL *does* specify how rendering windows are created and manipulated. IRIS GL's windowing interface is reasonably popular largely because it is simple to use. IRIS GL programmers can worry about graphics programming without needing to be an expert in programming the native window system. Experience also demonstrated that IRIS GL's windowing interface was high-level enough that it could be retargeted to different window systems. Silicon Graphics migrated from NeWS to the X Window System without any major changes to IRIS GL's basic windowing interface.

Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs. Unfortunately, the lack of a window system interface for OpenGL is a gap in

OpenGL's utility. Learning native window system APIs such as the X Window System's Xlib or Motif can be daunting. Even those familiar with native window system APIs need to understand the interface that binds OpenGL to the native window system. And when an OpenGL program is written using the native window system interface, despite the portability of the program's OpenGL rendering code, the program itself will be window system dependent.

Testing and documenting OpenGL's functionality lead to the development of the tk and aux toolkits. Unfortunately, aux has numerous limitations and its utility is largely limited to toy programs. The tk library has more functionality than aux but was developed in an *ad hoc* fashion and still lacks much important functionality that IRIS GL programmers expect, like pop-up menus and overlays.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Features from the IRIS GL, aux, and tk interfaces are included to make it easy for programmers used to these interfaces to develop programs for GLUT.

2.2.2.3 Design Philosophy (Kilgard, 1996)

GLUT simplifies the implementation of programs using OpenGL rendering. The GLUT application programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT API (like the OpenGL API) is stateful. Most initial GLUT state is defined and the initial state is reasonable for simple programs.

The GLUT routines also take relatively few parameters. No pointers are returned. The only pointers passed into GLUT are pointers to character strings (all strings passed to GLUT are copied, not referenced) and opaque font handles.

The GLUT API is (as much as reasonable) window system independent. For this reason, GLUT does not return *any* native window system handles, pointers, or other data structures. More subtle window system dependencies such as reliance on window system dependent fonts are avoided by GLUT; instead, GLUT supplies its own (limited) set of fonts. For programming ease, GLUT provides a simple menu sub-API. While the menuing support is designed to be implemented as pop-up menus, GLUT gives window system leeway to support the menu functionality in another manner (pull-down menus for example).

Two of the most important pieces of GLUT state are the *current window* and *current menu*. Most window and menu routines affect the *current window* or *menu* respectively. Most callbacks implicitly set the *current window* and *menu* to the appropriate window or menu responsible for the callback. GLUT is designed so that a program with only a single window and/or menu will not need to keep track of any window or menu identifiers.

This greatly simplifies very simple GLUT programs. GLUT is designed for simple to moderately complex programs focused on OpenGL rendering. GLUT implements its own event loop. For this reason, mixing GLUT with other APIs that demand their own event handling structure may be difficult. The advantage of a built-in event dispatch loop is simplicity.

GLUT contains routines for rendering fonts and geometric objects, however GLUT makes no claims on the OpenGL display list name space. For this reason, none of the GLUT rendering routines uses OpenGL display lists. It is up to the GLUT programmer to compile the output from GLUT rendering routines into display lists if this is desired.

GLUT routines are logically organized into several sub-APIs according to their functionality. The sub-APIs are:

Initialization. Command line processing, window system initialization, and initial window creation state are controlled by these routines.

Beginning Event Processing. This routine enters GLUT's event processing loop. This routine never returns, and it continuously calls GLUT callbacks as necessary.

Window Management. These routines create and control windows.

Overlay Management. These routines establish and manage overlays for windows.

Menu Management. These routines create and control pop-up menus.

Callback Registration. These routines register callbacks to be called by the GLUT event processing loop.

Color Index Colormap Management. These routines allow the manipulation of color index color maps for windows.

State Retrieval. These routines allow programs to retrieve state from GLUT.

Font Rendering. These routines allow rendering of stroke and bitmap fonts.

Geometric Shape Rendering. These routines allow the rendering of 3D geometric objects including spheres, cones, icosahedrons, and teapots.

2.3 Analysis Study

2.3.1 GridSET Laparoscopy Surgical training.

2.3.1.1 Background

The ***GridSET - VR simulator based training courses Training course for laparoscopic surgery*** aims at helping the trainee to become familiar with the initial steps of laparoscopic surgery, which are common in all laparoscopic operations, laparoscopic cholecystectomy being the prototype.

Unique, in comparison to other training approaches, is the possibility to interact with a 3-D simulated model. The model can be viewed from all sides and be made transparent. The procedure can be performed and an objective evaluation of the quality of the surgery is given to the trainee.

At the end of this GridSET course, the trainee will be capable to (Gridset Exchange, 2003):

- Choose the appropriate tool
- Choose the correct position for insertion of the Veress needle and the trocars
- Insert the Veress needle and the trocars accurately
- Establish an adequate pneumoperitoneum

In the end the trainee will have the appropriate skills to perform the initial steps of a laparoscopy operation.

The GridSET laparoscopy surgery training course is divided into four parts (Gridset Exchange, 2003):

- The theoretical basis of the laparoscopic approach
- Video clips
- Practice on the simulator
- Practice with multiple-choice questions.

Figure 2.3: GridSET Laparoscopy Surgical training snapshot.

Figure 2.3 shows the laparoscopy tools that the learner-surgeon uses in the laparoscopy simulation.

2.3.1.2 Product Specification

A GridSET Integrated training course is a complete training course that can be used in a web based class-environment. It contains (Gridset Exchange, 2003):

- A complete training course for laparoscopic surgery, with theory, examples, multiple choice questions and VR based simulator
- GridSET evaluation module: allowing students to assess their skills. Can also be used in examination mode
- GridSET conference server: allows the teacher and students to collaboratively view and work on the same surgery simulator. Comes with communication possibilities between participants.
- GridSET training server: a container to run one or more GridSET training courses.
- License for a maximum of 50 students

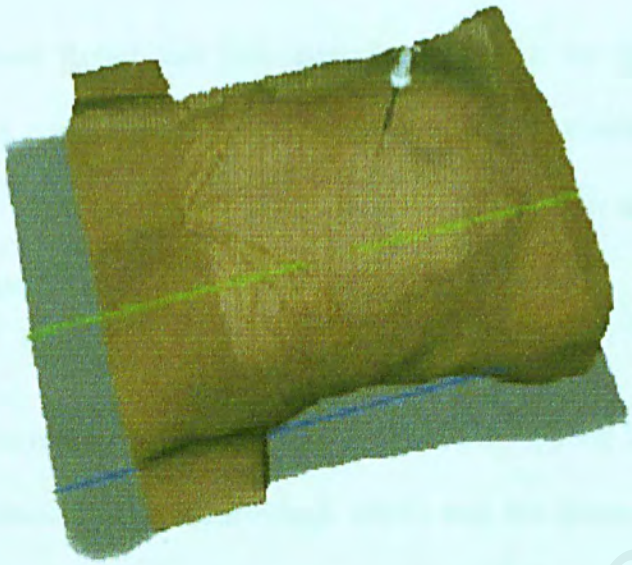

Figure 2.4: GridSET Laparoscopy Surgical training snapshot.

Figure 2.4 shows the Verres needle, one of the tools used for the laparoscopic surgery, inserted into the patient

2.3.2 ARTEMIS

2.3.2.1 Background

ARTEMIS (Advanced Robot and Telemanipulator System for Minimal Invasive Surgery) features a complete telepresence system which allows the surgeon to perform minimally invasive surgery (MIS) remotely via a man machine interface with multimedia capabilities.

ARTEMIS is a joint research project of the Institute for Applied Informatics (IAI), the former Hauptabteilung Ingenieurtechnik (HIT) and the former Hauptabteilung Versuchstechnik (HVT) of the Forschungszentrum Karlsruhe and the Universitätsklinikum Tübingen (Institute for Applied Informatics (IAI), 2003).

Figure 2.5: The ARTEMIS system.

Figure 2.5 shows the experimental operating theatre for minimally invasive surgery with the MMI in front and the operating table with the work system in the back.

2.3.2.2 Product Specification

The ARTEMIS system consists of the following components (Institute for Applied Informatics (IAI), 2003):

- Man Machine Interface (MMI):
    - Graphical user interface; 3D video imaging of the operating environment; simulation system KISMET
    - Master
    - Other input units: Speech input; foot pedal; trackball

Figure 2.6: Man Machine Interface

Figure 2.6 shows the ARTEMIS user interface system with two identical HIT-Master-II, and three video screens for the graphical user interface, the 3D video endoscope picture and the simulation

- Work system:

  ARTEMIS currently has two different telemanipulation units:

    - TISKA

      a computer controlled carrier system with surgical effectors.

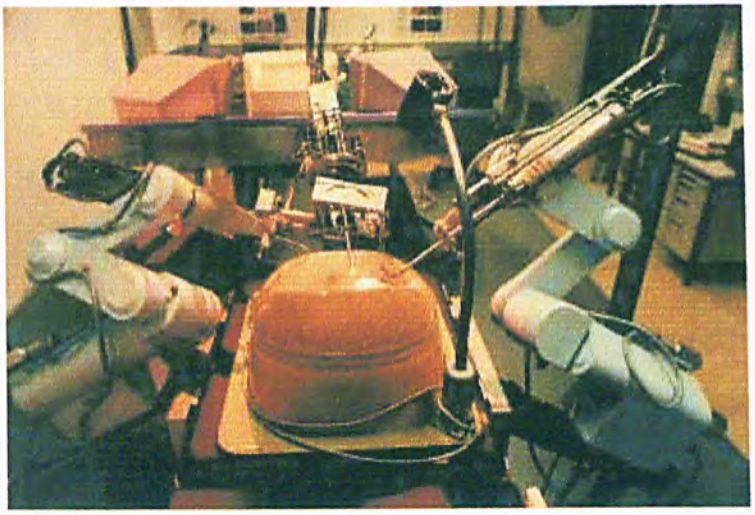

Figure 2.7: TISKA

Figure 2.7 shows an artificial body (Pelvitrainer) with two IGSs TISKA and in the middle the EGS ROBOX.

- ROBOX

  a computer controlled endoscope guidance system.

- Control system:
  - MONSUN
  - KISMET
- MoMo Software Concept:

  Distributed software concept for real-time systems.

2.3.2.3 KISMET

The KISMET (Kinematic Simulation, Monitoring and Off-Line Programming Environment for Telerobotics) software is under development at Forschungszentrum Karlsruhe since 1986 as a 3D realtime simulation support tool for (The KISMET 3D, 2003):

- effective planning, simulating, programming and monitoring of teleoperation tasks
- robotics simulation
- high-performance graphical visualisation
- medical simulation
- Virtual-Reality (VR) application kernel
- scientific visualisation

KISMET features (The KISMET 3D, 2003):

- Hierarchical data concept for interactive detail selection
- Sensor based synthetic viewing (Virtual Reality)
- Kinematic simulation of complex mechanisms
- Realtime multibody-system dynamics simulation
- Realtime elastodynamics simulation, tissue deformation
- High-quality rendering of polygonal, NURBS and voxel-volume geometry data models
- Camera simulation, model based control and target tracking
- Robot off-line programming and simulation
- Stereo viewing (shutter glasses)
- CAD data import tools (STEP, DXF, VDA-FS, Wavefront, SoftImage ...)

- KISMET is implemented in C, as graphics interface GL is used
- KISMET is available for SILICON GRAPHICS (TM) workstations

2.3.3 VEST System *One* (VS*One*)

2.3.3.1 Background

"Virtual Endoscopic Surgery Training" (VEST) system VS*One* was developed within the framework of the joint TT-project (technology transfer) "LAParoscopy-SIMulator" (internal project name, not a product name) by the partners Forschungszentrum Karlsruhe / Institut für Angewandte Informatik (research group: "VR-Systems/Realtime-Simulation" - contact: Dr. Uwe Kühnapfel) and the company "Select IT VEST Systems AG" / Bremen.

The VS*One* system is commercially available through "Select IT VEST Systems AG" (Virtual Endoscopic Surgery Training, 2003).

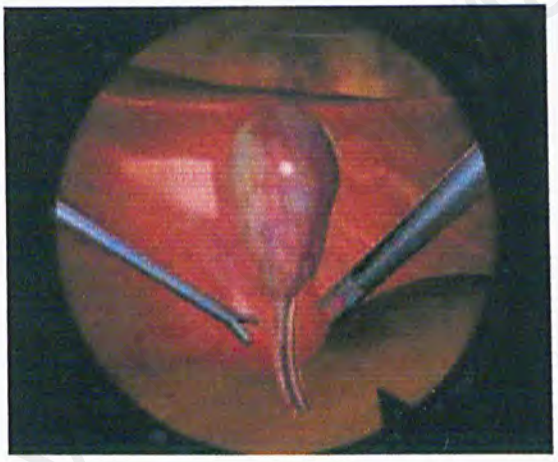

Figure 2.8: Simulation Scene "Laparoscopic Cholecystectomy"

Figure 2.9: Series System VSOne, with fiber housing box

Figure 2.10: Haptic instrument interface

2.3.3.2 Features

VEST System **VS*One*** features (VEST System *One* Technology, 2003):

- new compact design with electrically driven trainer input box (instrument box) to ensure an optimal working position and optimum storage size
    - 3 haptic (force-feedback) devices as mock-up endoscopic instruments
    - 1 virtual endoscopic camera
- three new ***Basic Task Training (BTT)*** exercises
    - find numbers in tubes (training to handle the endoscopic camera)
    - touch points on blocks (left/right hand basic instrument handling and 3D-positioning)
    - follow a prescribed path (left hand: grasp a plate with a figure; right hand: follow the figure with a pencil)
- improved laparoscopic cholecystectomy scenario
- improved laparoscopic gynaecology scenario

2.3.4 LAP Mentor

2.3.4.1 Background

The ***LAP Mentor***™ multi disciplinary simulator enables hands on practice for a single trainee or a team at any given time. The system offers opportunities for perfecting basic generic laparoscopic skills and for performing complete laparoscopic surgery procedures.

Figure 2.11: LAP Mentor Simulation Scene

2.3.4.2 Features

As with all simulators in the Simbionix line of medical training simulators, the ***LAP Mentor***™ is designed to meet the training and practice needs of physicians and the evaluation and management needs of their instructors. The following features are an integral part of the simulator (Simbionix, 2003):

- **Realism** - realistic visualization of the human anatomy that provides a life-like view of the intraabdominal cavity. One can maneuver the gallbladder, expose the cystic duct and artery, clip and cut the cystic duct and artery and separate the gallbladder from the liver with electrocauterization.

- **Tactile sensations** - felt in the use of laparoscopic instruments that imitate real-life.

- **Anatomical variations** – virtual patient cases include a variety of individuals of both sexes with varying internal anatomies.

- **Basic Tasks and full procedures** - opportunities to practice in order to gain basic laparoscopic surgery skills as well as practice on full procedures on virtual patients.

- **Management mode** - for organization of trainees, courses and workshops. The data collection and exporting features of this mode facilitate research work as well.
- **Extensive evaluation parameters** for each performance on the simulator.
- **Variety of educational aids**

CHAPTER 3 METHODOLOGY

3.0 Methodology

3.1 Overview

Methodology was defined as a collection of procedures, techniques, tools, and paradigm.

The goal of any methodology is to provide a quality and cost-effective system. Methodology provides a framework for control and coordination of the development effort. No single methodology is "best" under all or even many situations.

System development methodology is a method to create a system with a series of steps and operations or can be defined as a system life cycle model. Every system development process model includes system requirements such as user, needs, resource as input and a final product as output.

There are several process models in system development which includes

- Waterfall Model
- Spiral Model
- Rapid Application Development Model

3.2 Spiral Model

3.2.1 Introduction

The spiral lifecycle model is the combination of the classic waterfall model and an element called risk analysis (Spiral Lifecycle, 2003). This model is very appropriate for large software projects. The model consists of four main parts, or blocks, and the process is shown by a continuous loop going from the outside towards the inside. This shows the progress of the project.

- Planning

  This phase is where the objectives, alternatives, and constraints are determined.

- Risk Analysis

  What happens here is that alternative solutions and constraints are defined, and risks are identified and analyzed. If risk analysis indicates uncertainty in the requirements, the prototyping model might be used to assist the situation.

- Engineering

  Here the customer decides when the next phase of planning and risk analysis occur. If it is determined that the risks are to high, the project can be terminated.

- Customer Evaluation

  In this phase, the customer will assess the engineering results and make changes if necessary.

Navigation between each phase is done through data/control hierarchies, functional decomposition, and requirement allocation.

Phase products are represented through structure charts, state-transition diagrams, and stimulus response threads.

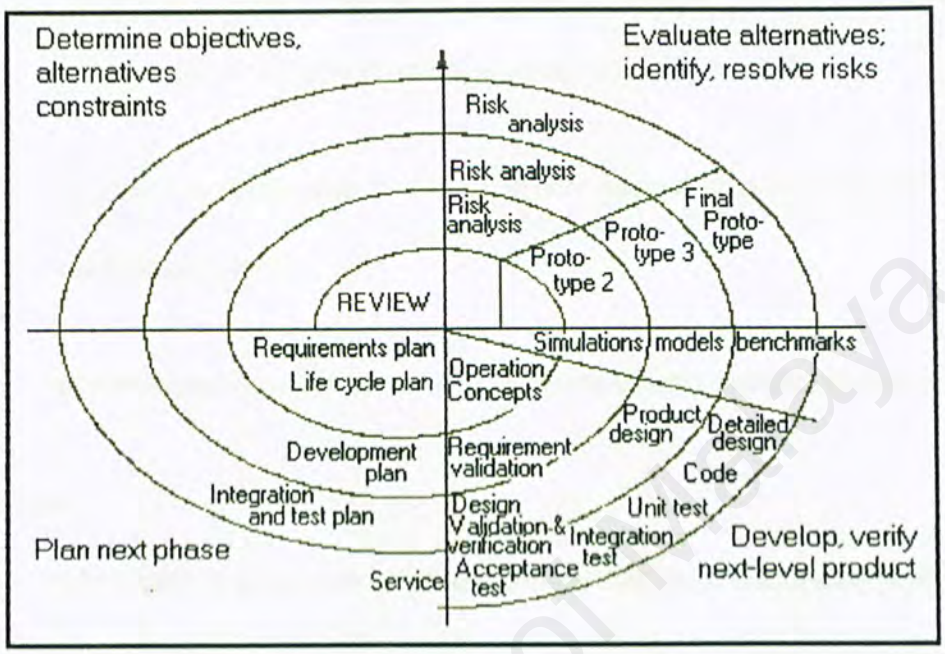

Figure 3.1: The Spiral Model

3.2.2 Strengths and Weaknesses

Strengths:

- Good for large and complex projects
- Customer Evaluation allows for any changes deemed necessary, or would allow for new technological advances to be used
- Allows customer and developer to determine and to react to risks at each evolutionary level
- Direct consideration of risks at all levels greatly reduces problems

Weaknesses:

- Difficult to convince some customers that the evolutionary approach is controllable
- Needs considerable risk assessment
- If a risk is not discovered, problems will surely occur

3.2.3 Appropriate or Inappropriate Domains of Application

Appropriate:

- Large, complex projects

Inappropriate:

- Simple, easy projects

  If a project is simple and easy, then time will be wasted on risk analysis as the risk could be easily seen or there weren't any risks at all.

3.2.4 Comparison

The Spiral model is actually based in part on the Waterfall model (the other part is based on the Rapid Prototype model). The Spiral model is better in the sense that it allows for risk management where the Waterfall places too much emphasis on project management. It should be kept in mind however, the requirements of the project, and which model will suit it the best.

3.3 Rapid Application Development Model

3.3.1 Introduction

Rapid Application Development can be defined as a software development process that allows usable systems to be built in as little as 60-90 days but often with some compromises.

According to Rapid Application Development (2003),

> In certain situations, a usable 80% solution can be produced in 20% of the time that would have been required to produce a total solution. The business requirements for a system can be fully satisfied even if some of its operational requirements are not satisfied. In addition, the acceptability of a system can be assessed against the agreed minimum useful set of requirements rather than all requirements.

3.3.2 Strengths and Weaknesses

Strengths:

- Buying may save money compared to building
- Deliverables sometimes easier to port because they make greater use of high-level abstractions, scripts, intermediate code
- Development conducted at a higher level of abstraction because RAD tools operate at that level
- Early visibility because of prototyping
- Greater flexibility because developers can redesign almost at will
- Greatly reduced manual coding because of wizards, code generators, code reuse
- Increased user involvement because they are represented on the team at all times
- Possibly fewer defects because CASE tools may generate much of the code
- Possibly reduced cost because time is money, also because of reuse
- Shorter development cycles because development tilts toward schedule and away from economy and quality
- Standardized look and feel because APIs and other reusable components give a consistent appearance

Weaknesses:

- Buying may not save money compared to building
- Cost of integrated toolset and hardware to run it
- Harder to gauge progress because there are no classic milestones
- Less efficient because code isn't hand crafted
- Loss of scientific precision because no formal methods are used
- May accidentally empower a return to the uncontrolled practices of the early days of software development
- More defects because of the "code-like-hell" syndrome
- Prototype may not scale up
- Reduced features because of time boxing, software reuse
- Reliance on third-party components may

    1. sacrifice needed functionality
    2. add unneeded functionality
    3. create legal problems

- Requirements may not converge because the interests of customers and developers may diverge from one iteration to the next
- Standardized look and feel (undistinguished, lackluster appearance)

• Successful efforts difficult to repeat because no two projects evolve the same way

• Unwanted features through reuse of existing components

3.3.3 Appropriate or Inappropriate Domains of Application

Appropriate:

• The application will be run standalone.

• Major use can be made of preexisting class libraries (APIs).

• Performance is not critical.

• Product distribution will be narrow (in-house or vertical market).

• Project scope (macro-schedule) is constrained.

• Reliability is not critical.

• System can be split into several independent modules.

• The product is aimed at a highly specialized IS (information systems) market.

• The project has strong micro-schedule constraints (timeboxes).

• The required technology is more than a year old.

Inappropriate:

• Application must interoperate with existing programs.

- Few plug-in components are available.

- Optimal performance is required.

- Product development can't take advantage of high-end IS tools

- Product distribution will be wide (horizontal or mass market).

- RAD becomes QADAD (Quick And Dirty Application Development).

- RAD methods are used to build operating systems (reliability target too high for RAD), computer games (performance target too high for RAD).

- Technical risks are high due to use of "bleeding" edge technology.

- The product is mission- or life-critical.

- The system cannot be modularized (defeats parallelism).

3.4 Waterfall Model

3.4.1 Introduction

The Waterfall Model is the most commonly used approach for major acquisition systems over the past several decades. Under this approach there are a series of steps that will have to be achieved from system concept to system operations and all will be preformed in series, not parallel. The transition from each step is only accomplished after successful completion of a very structured review process. The following lists the process step and corresponding review for each phase:

Task/Step	Review
1. Requirement Definition	1. System Requirement Review
2. Analysis	2. Risk Assessment Review
3. Design	3. Preliminary/Critical Design Reviews
4. Coding	4. Walk Through Review
5. Testing	5. Technical Evaluation Review/Operational Evaluation Review
6. Operations	6. Initial Operational Capability

For each task and review there are many structured documents that are prepared, reviewed, and maintained for the life of the system. The highly structured nature of the waterfall method makes it quite applicable for large well-defined projects.

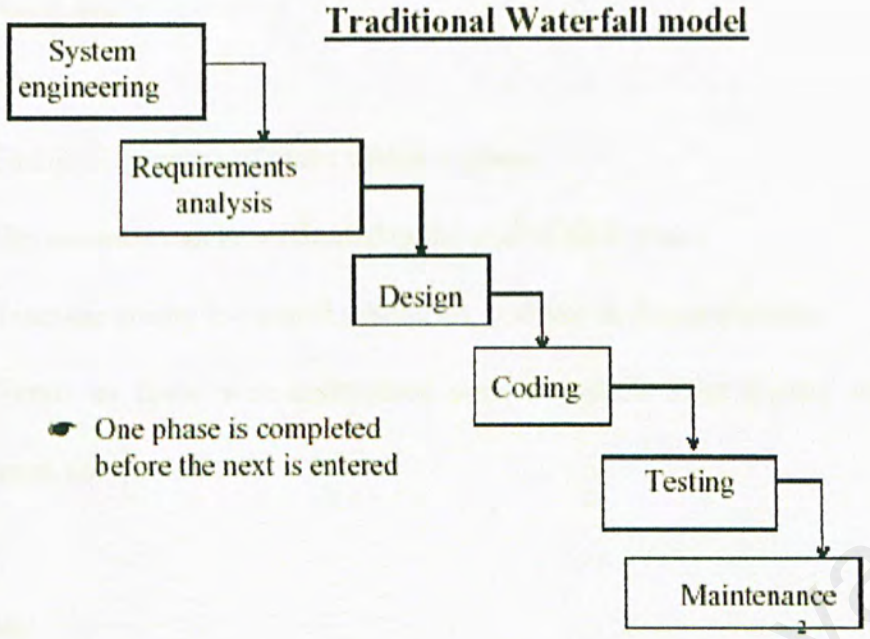

Figure 3.2: The Traditional Waterfall Model

3.4.2 Strengths and Weaknesses

Strengths:

- Enables allocation of tasks within a phase.
- The progress can be evaluated at the end of each phase.
- It has the ability to control schedules, budgets & documentation.
- Tends to favor well-understood system aspects over poorly understood system components.

Weaknesses:

- Projects rarely flow in a sequential process.
- Difficult to define all requirements at the beginning of a project.
- Unresponsive to changes.
- A working version of the system is not seen until late in the project's life.
- Repairing problems further along the lifecycle becomes progressively more expensive.
- Maintenance costs can be as much as 70% of systems costs

3.5 Prototyping

Prototyping is a sub-process and a prototype is a system or partially complete system that is built quickly to explore some aspects of the system requirements. It is constructed with various objectives.

Benefits of prototyping

- To ensure that the system meets the performance goals and constraints
- To ensure the system are practical and flexible
- To ensure the system fulfills the users' requirements
- To have an insight of how the module and sub-modules interact with each other

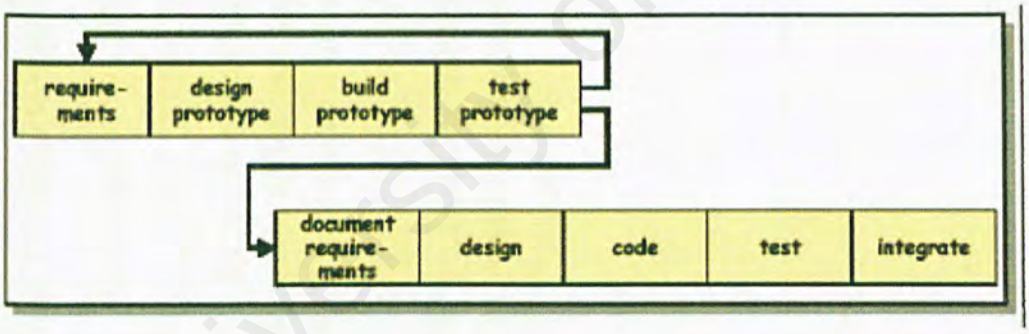

Figure 3.3: Prototyping Lifecycle

3.6 Comparison

Table 3.4: Various Methodology's Comparison

Waterfall Model	Rapid Application Development Model	Spiral Model
Pro. 👍 Documentation 👍 Maintenance easier 👍 Quality product at finish	Pro. 👍 Allows frequent changes 👍 Helps define user requirements 👍 Rapid return on investment	Pro. 👍 Risk analysis preceding each phase 👍 Allows for changing requirements 👍 Allows prototyping
Con. 👎 Specification document 👎 Have to get it right first time 👎 Does not allow for prototype 👎 Time consuming 👎 Costly 👎 Hard to accommodate 👎 Doesn't accommodate new requirements	Con. 👎 Increased maintenance costs 👎 How do you know you are finished? 👎 Build-and-fix	Con. 👎 Once risk cannot be mitigated the project is terminated. 👎 Not effective for large-scale projects.

3.7 Methodology Chosen

The Waterfall Model with Prototyping was chosen because

- A good specification to begin with
- Easy to use
- Systematic
- Scope of project well understood
- Project risks have been accessed and are considered to be low

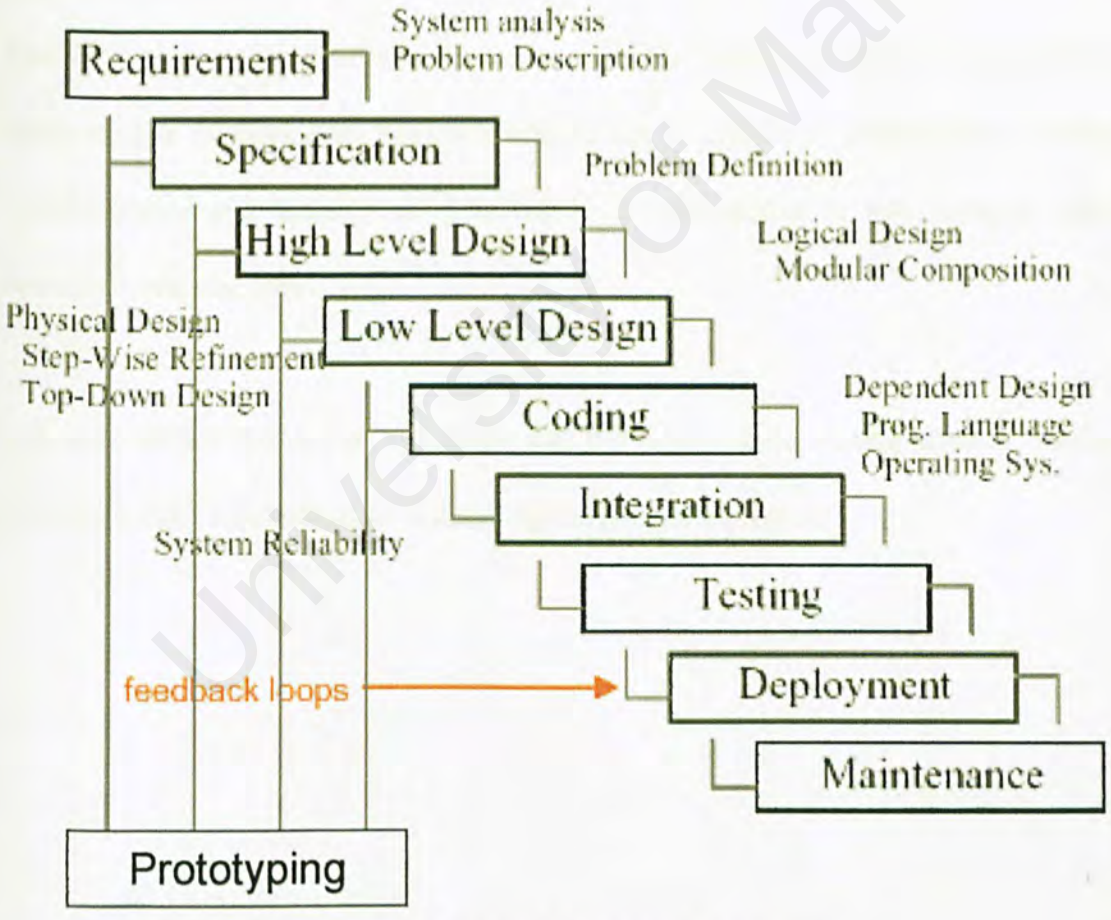

Figure 3.5: Waterfall Model with Prototyping

3.7.1 Process Steps and Corresponding Descriptions

Requirements

Requirements are statements of functions and behavior of the system required by its users and operators. Most general requirements define broad & detailed objectives of the system which includes reliability, correctness, efficiency, user-friendliness and expandability. In addition, requirements also outline the relationship of qualitative and quantitative system goals.

Specification

Specification is a listing of specific and measurable behavioral system constraints that satisfy system requirements. Specifications should be complete, unambiguous, minimal, understandable and testable as it serves as a communication link between system operations and end users.

It is also defines the design validation and the final system testing criteria. Finally it provides a chief mechanism for estimating the project's progress.

Design: Representation or model of a system

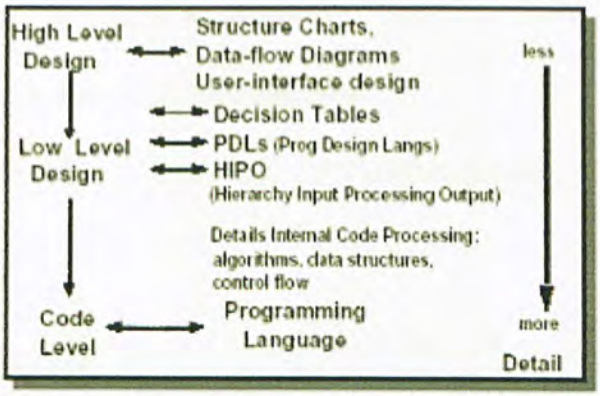

Figure 3.6: Design Flowchart

Prototyping

Prototyping is used for:

- understanding the requirements for the user interface
- examining feasibility of a proposed design approach
- exploring system performance issues

One of the problems of prototyping is that users often treat the prototype as the solution. A prototype is only a partial specification of the solution.

Coding and Debugging (implementation)

Coding and Debugging process is the translation of system design into a programming language.

Functionality of the Program Unit Notebooks

1. Documents programmer's work activities

2. Maintains current unit (module) documentation

3. Passed from programmer to programmer during development

Unit Name:_____ Programmer:_____
Routines Included:_____

SECTION	CONTENTS	DUE DATE	COMPLETED DATE	REVIEWER/DATE
1.	RQMTS.			
2.	ARCH. DESIGN			
3	DETAIL DESIGN			
4.	TEST PLAN			
5.	TEST RESULTS			
6.	CHANGE REQUESTS			
7.	SOURCE CODE			
8.	NOTES			

RELEASE APPROVAL:_____ DATE:_____

Figure 3.7: An example of a Program Unit Notebook

The Data Dictionary records information and physical format details of all structures, variables and files.

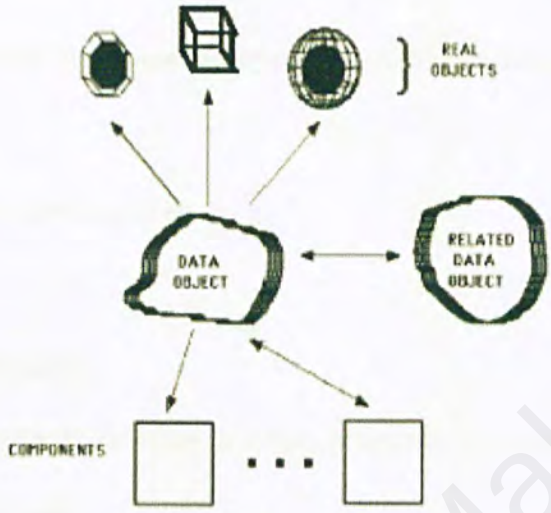

Figure 3.8: Data Dictionary Flowchart

Data Dictionary Entry

Name : from the data-flow diagram or structure chart

Routine Usage: routines that access the object

Purpose : explanation

Derivation : where the data that the items hold comes from

Sub-items : Record components

Notes : comments

Integration and Testing

Integration and Testing are divided into two

1. Unit testing

   Individual modules (functions) are tested separately from other modules.

2. Integration testing

   System modules are tested together.

Deployment and Maintenance

Deployment and Maintenance requires previous phases to be repeated. It makes up 70%-90% of the total system cost.

The majority of maintenance time (50%) is spent on system understanding which includes system documentation.

Maintenance Tasks

- collection, analysis and prioritization of user trouble reports
- new system release installations
- documentation (user's manuals) changes
- configuration control issues

CHAPTER 4 SYSTEM ANALYSIS

4.1 REQUIREMENT ANALYSIS

4.2 AUTHORING TOOLS

4.3 DEVELOPMENT TOOLS CHOSEN

4.4 SYSTEM REQUIREMENTS

4.0 System Analysis

4.1 Requirement Analysis

4.1.1 Techniques Used To Define Requirements

This section discusses the appropriate techniques that are used to define and elicit user's requirements.

Internet Research

80 % of this paper was written based on Internet research and findings.

Library Research

Medical journals and reports also proved to be a vital source of information in defining user's requirements.

4.1.2 Functional Requirements

This section will carefully assess the needs that this project needs to fulfill and how the project will be constructed.

Below is the listing of functional requirements that are deemed relevant to this project.

4.1.2.1 Instrument Insertion and Removal

The simulator will support the insertion (and removal) of any instrument into any cannula by the user. The following conditions or requirements apply to instrument insertion and removal:

a) Instrument detection is automatically performed by the simulator upon instrument insertion.

b) When the laparoscope is inserted, an iris like image will be generated as the simulated camera view travels through the cannula tube and opens up into the body cavity. When the laparoscope is removed, the opposite iris effect occurs.

c) If any tissue is being grasped in an instrument and that instrument is removed from the cannula, the tissue is discarded from the database. This, in effect, simulates the biopsy of small tissue elements as they are assumed to be removed from the instrument when it is retracted from the body.

d) Unless otherwise directed by the training instructor, the stapler instrument is assumed to be automatically reloaded with a staple cartridge if it is removed from the body.

Unless otherwise directed by the Training Instructor, the clip applier instrument is assumed to be automatically refilled with clips if it is removed from the body and is empty upon removal.

4.1.2.2 Tissue Deformation, Translation, Rotation

The simulator will support tissue deformation from interaction with surgical instruments. It will also support large-scale translation and rotation of tissue elements.

The individual tissue elements in the simulation are connected by springs to allow the organs to maintain an expected shape, but still allow for movement and deformation.

Tissue is deformed in several ways:

a) One or more nodes are grasped in a surgical instrument. Subsequent instrument movements displace the grasped nodes, forcing the connecting nodes to stretch or contract. The stretch forces affect neighboring nodes, allowing for the force and motion to be properly distributed throughout the tissue.

b) Nodes come in contact with a surgical instrument. This causes a collision handling process to govern the forces on the affected nodes to move them with the instrument to prevent the instrument from moving through the tissue.

c) Nodes come into contact with other nodes (i.e., tissue contacts tissue). This is handled in the same manner as an instrument-to tissue collision.

d) Other external forces are applied. These forces include:

i) Simulated pulsing of large blood vessels

ii) Simulated diaphragm motion

iii) Local alterations in tissue properties caused by cautery, laser burning, etc.

iv) Constraints imposed by surgical staples, sutures, etc.

v) Change in abdomen pressure caused by insuflation or desuflation

Since the tissue elements are grouped to form structures (i.e., organs), sufficient force can be applied to a group of nodes such that the resultant motion affects *all* nodes in the structure. At this point, the structure can be translated and / or rotated as a structure. For example, the gall bladder can be lifted and moved within the body.

Deformation can also stretch the tissue links to the point at which they break and cause tissue tearing. This requirement is detailed in a following section.

4.1.2.3 Tissue Cutting, Tearing, and Puncture

The simulator will support tissue separation from cutting and from tearing. It will also support puncture of tissue by rigid instruments.

The simulated nodes in the tissue element models can be stretched to deform the tissue shape. Should the force on these spring links become excessive, the link will shear, simulating tissue tearing.

Cutting occurs when an instrument defined as having a sharp surface comes in contact with tissue and a force is applied. As the sharp edge moves into or over a tissue surface, the normal link breaking thresholds are greatly reduced or eliminated. This allows tissue tearing to occur with only a slight application of force.

After separating, the manner in which the tissue was separated determines how the opening is handled by the models. The separation can either split tissue elements at their joints (provided that the cut or tear was sufficiently near a joint) or cause the elimination of a tetrahedral tissue element. In this case, the neighboring points will be altered such that the removal of the tetrahedron does not appear too drastic a change.

If the tissue was torn, the neighboring points will be moved somewhat randomly or roughly to 'close up the gap'. For an incision caused by a sharp instrument, a cleaner adjustment is made.

The dark triangles in the original surface patch represent triangles that are to be removed because of a cut or tear. After a 'clean' cut, the neighbor vertices are adjusted to form a regular incision in the tissue. However, if the triangles were removed due to a tear, the resulting vertex moves leave an irregular tear in the tissue. To simulate the propagation of a cut or tear, the breaking thresholds of neighboring tissue elements can be reduced.

Punctures occur when a rigid, tubular instrument makes contact with tissue along the instrument's axis. In this case, it is required that the interaction be handled differently than a cut or tear. For example, if a needle or blunt probe is pressed against tissue with enough force to puncture it, it can then be removed without leaving a long incision as would be occur for a cut.

Incisions, cuts, and tears will instantiate bleeding, depending on the tissue type.

4.1.2.4 Surgical Stapler

The system will support the simulation of a surgical stapler.

The surgical stapler is used to staple a section of tissue together to prevent bleeding when the tissue is cut. The cutting function is performed by a secondary function of the same instrument. After cutting, the bleeding should be minimal due to the staples. Cut tissue will resemble the following:

The following conditions are required for a successful staple and cut to take place:

a) An unused staple cartridge must be present in the simulated instrument (i.e., an empty or malfunctioning stapler is not being simulated)

b) Tissue must be fully within the staple area of the stapler instrument tip (any tissue not touching the staple area will not receive a staple when the instrument is activated)

c) The instrument's 'staple' activation handle must be fully actuated (partial actuation may result in some or all staples being only partially implanted into the tissue)

d) The instrument's 'cut' activation handle must be fully actuated (partial actuation may result in a shorter cut)

A successful staple and cut will result in the following:

a) Adjustment of the texture of the stapled tissue to show the staples. These may also be handled as individual rigid objects in the force and visual models if possible, to allow them to be removed or touched, providing a force cue.

b) Adjustment of the bleeding generation algorithm for the stapled tissue elements. Normally, the cut face would bleed in accordance with the blood generation

requirements listed in this document. Properly stapled tissue would cause a reduction in the blood generation.

4.1.2.5 Surgical Clip Application

The system will support the simulation of a surgical clip application.

The surgical clip applier is used to place a metal clip around a piece of tissue, typically a tubular structure, prior to cutting. The clip prevents bleeding from the cut vein or fluid flow from a cut duct.

The following conditions are required for a successful clip application to take place:

a) The simulated clip applier must contain at least one clip (i.e., an empty or malfunctioning clip applier is not being simulated)

b) The tissue must be fully within the clip area of the instrument tip (if any tissue is not fully within the clip area, a partial application can result)

c) The instrument's activation handle must be fully actuated (partial actuation may result in a partial application or allow the clip to fall off after application; additionally, the clip may puncture the tissue and become embedded in it)

A successful clip application will result in the following:

a) The addition of a clip object into the database. This may be handled as a discrete rigid object or as a combined force model and visual effect.

b) The clip object will compress the associated tissue, typically closing off a tubular object. This will serve to reduce or prevent fluid flow when the object is cut.

4.1.2.6 Suction

The system will support the simulation of a vacuum suction instrument.

The suction function is part of the dual-function suction / irrigation instrument. Suction is used to remove pooled fluid (saline, blood, etc.) from the surgical area and also remove accumulated smoke from cautery.

The following conditions are required for suction to take place:

a) The simulated suction / irrigation instrument must be functioning (i.e., a disconnected or blocked suction pump is not being simulated)

b) The portion of the suction area of the instrument tip that is fully submerged in liquid will determine the rate and amount of liquid removal that occurs. The portion of the suction area that is exposed to air will determine the amount of air / gas / smoke removal that occurs.

c) The rate of removal of both liquid and gas / smoke is also proportional to the amount of actuation the surgeon applies to the facsimile instrument's suction valve button.

d) The instructor also has control over the suction function, which can be enabled, disabled, applied, or adjusted. If the insuflation pump cannot maintain the required flow, the use of the suction instrument to remove gas or smoke can cause desuflation of the abdomen.

4.1.2.7 Irrigation

The system will support the simulation of an irrigation instrument.

The irrigation function is part of the dual-function suction / irrigation instrument. Irrigation is used to wash blood and debris from an incision or from the laparoscope lens. Saline is used as the irrigation fluid.

The following conditions are required for irrigation to take place:

a) The simulated suction / irrigation instrument must be functioning (i.e., a disconnected or blocked saline source is not being simulated)

b) The rate of liquid flow from irrigator tip is proportional to the amount of actuation the surgeon applies to the facsimile instrument's irrigation valve button.

c) The actual liquid flow path from the irrigator is handled in the same manner as bleeding. This requires that the fluid travel in an arc until it contacts a surface, it will flow along the surface, and eventually pool in the body until removed via suction.

d) The instructor also has control over the irrigation function, which can be enabled, disabled, applied, or adjusted.

e) Irrigation liquid flow will perform the following functions:

i) It will reduce the simulated friction of any tissue element that it makes contact with (to a predetermined limit, i.e., it cannot make tissue that is already wet any 'wetter'). This affects the simulated stiction; by reducing stiction / friction, small tissue particles will tend to be 'washed away' by saline flow.

ii) It will reduce the coloring effect of any tissue that is currently marked as having blood on it. If a tissue element has blood on it, continued saline flow over the tissue will clear the blood from the tissue over several frames. The same effect applies to the laparoscope lens when it is dirty. There is a minimum

threshold after which the tissue is set as having no blood on it at all. This allows a surface to be completely 'cleaned' by saline wash.

iii) If a tissue element is generating blood, the wash of saline over that tissue will cause the resulting blood to be less red as it flows over other tissues.

4.1.2.8 Bleeding

The system will support bleeding from the appropriate tissue sources.

There are two types of bleeding that will occur in the simulation: high-pressure bleeding (spurting from a cut vein or artery), and low pressure bleeding (seeping from smaller sources such as capillaries when an incision is made).

In this case, the blood source would be the source of high-pressure bleeding such as an artery. When cut, several factors would determine the blood arc trajectory which is the initial path that the blood will take when traveling through the cut. When this stream makes contact with any other surface, blood flow paths (or paths) are computed to allow the blood to flow along the surface towards ground. As the liquid reaches the lowest level in the body, it begins pooling and raises the blood pool level.

4.1.2.9 High-Pressure Bleeding

High-pressure bleeding occurs when tissue is cut, torn, or weakened near an area defined as a high-pressure blood source.

Typically, this will be a larger vein or artery where such a rupture will cause a large amount of blood to enter the surgical area. Depending upon the size of the opening and the simulated pressure, the blood will spurt at the current simulated heart rate and cause the flow to arc as it falls.

The following factors influence the generation of high-pressure bleeding:

a) The tissue type at the incision and associated simulated blood source (i.e., a major vein, a minor vein, partially cauterized, etc.).

b) The size of the rupture or orifice through which the blood must flow.

c) A nick in the wall of an artery can rupture on its own due to the loss in wall thickness. Therefore, the blood generation routines must process any cutting or tearing on this type of tissue even if a complete opening has not been made. If the algorithm determines that the wall can self-rupture, it will instantiate an opening, after which the other blood flow routines can process the actual flow through the opening.

d) The above rupture algorithm can also enlarge an existing opening based on tissue type, thickness, and blood pressure.

e) If the blood generation is from a large vein or artery, the blood pressure will not be simulated as a constant, but more appropriately will pulse from low-to-high-to low values. This pressure waveform will enter the generation routine, causing a cyclic change in the flow and arc results.

4.1.2.10 Low-Pressure Bleeding

Low-pressure bleeding occurs when tissue is cut, torn, or weakened near an area that is not defined as a high-pressure blood source. Typically, this will be most tissue, fat, and

the like where the blood flow is from capillaries instead of large-diameter veins or arteries.

Incisions or tears in this type of tissue generate blood generation and seepage, instead of the generally directed spurt or flow from a high-pressure bleed.

The following factors influence the generation of low-pressure bleeding:

a) The tissue type at the incision (i.e., fatty tissue, liver, etc.)

b) The manner that it was cut (incision, tear, burn, etc.)

c) The size of the exposed bleeding surface

d) The amount of cautery present

As blood is generated from this tissue, it will tend to flow towards ground. As it reaches the base of the body cavity, it will begin to pool. This pooled blood can be removed via suction.

Blood that was generated from a high-pressure bleed and has landed on another tissue surface will use the flow equations to create a path to ground.

4.1.2.11 Liquid Pooling

The simulator will support pooling of liquids such as blood or saline and the reduction of the pool by suction.

Liquid pooling occurs when any liquid such as blood or saline flows until it reaches a tissue cavity capable of containing an amount of liquid. Currently, only a single liquid

pool level is maintained in the simulator. In effect, this allows all liquid to flow to the base of the body cavity, at which point it will begin to collect and produce a pooled liquid height.

Blood flow paths along the organs eventually reach the 'floor' of the database area. At this point, all liquid flows contribute to integrating the pool level. The surface of this pool is shown by the visual system.

The color and transparency of this pool surface are determined by the portions of blood and saline that comprise the pooled liquid (the individual fluid types that comprise the pool are maintained separately and added to form the total pool level). Suction is used to reduce (and potentially eliminate) the pooled liquid.

4.1.3 Non-Functional Requirements

Non-Functional Requirements of a system are attributes and characteristics of the system.

4.1.3.1 Types of Non-Functional Requirements

i) Product-Oriented Attributes

• Performance —throughput, response time

• Usability—easy to learn and use

• Efficiency —minimal use of computing resources

• Reliability—long mean-time between failures; availability, correctness

• Security— prevents unauthorized access to programs and data

• Robustness—works in the presence of invalid input, faults, and stressful conditions

• Adaptability—reusable in other environments, for other problems

• Scalability—works with large data sets

• Cost —cheap to buy, install, and operate

ii) Family-Oriented Attributes

• Portability—easily modified to work on different platforms

• Modifiability— easily extended with new features

• Reusability—reuses design and components in other systems

iii) Process-Oriented Attributes

• Maintainability—easily modified (fixes, extensions)

• Readability— documents and code easy to read and understand

- Testability—easy to verify that product meets its specifications
- Understandability—design, architecture, and code are easy to learn
- Integratability—easy to integrate components; interoperability
- Complexity— level of interaction among modules

4.1.3.2 Selected Non-Functional Requirements

This project will not attempt to incorporate all of the above non-functional requirements but instead will concentrate on the selected characteristics that are deemed relevant.

Usability

The developed system must include ease-of-use, learnability, memorability, and a good User Interface Design in its features.

Efficiency

Efficiency refers to the level at which a software system uses scarce computational resources, such as CPU cycles, memory, disk space, buffers and communication channels.

The system must be able to aim at the minimal use of computing resources.

Reliability

The developed system must have the ability to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended.

Modifiability

The developed system must have the ability to add (unspecified) future functionality.

4.2 Authoring Tools

This section discusses the various authoring tools that are available for this project's development.

4.2.1 Microsoft Visual C++ .Net 2003 (Microsoft Corporation, 2003)

Microsoft Visual C++ .NET 2003 is a powerful tool for creating Microsoft Windows®-based and Microsoft .NET-connected applications, dynamic Web applications, and XML Web services using the C++ development language.

This robust development environment comprises compilers that are highly-conformant to the International Standards Organization (ISO), a Standard Template Library (STL) implementation, industry-standard Active Template Library (ATL) and Microsoft Foundation Class (MFC) libraries, and powerful integrated development environment (IDE) features enabling efficient editing and debugging of source code.

Features

- Able to create highly tuned .NET-connected applications and components
- Able to create highly tuned unmanaged Windows-based applications and components
- Able to move existing C++ code to .NET granularly and at a self-defined pace
- Able to build modern C++ code and library sources with a highly ISO C++ compliant compiler
- Able to utilize enhanced libraries to incorporate advanced features

- Advanced compiler and language features make writing complex code easier and safer

- Able to write code efficiently in an extensible IDE

- Able to debug and profile applications quickly and efficiently

4.2.2 3ds max™ 5 (Discreet, 2003)

3ds max, the world's most widely-used 3D modeling, animation, and rendering software, contains the essential high-productivity tools required for creating eye-catching animation, cutting-edge games, and distinct design visualizations. Version 5 raises the bar with some great new features and highly optimized workflows that will enable you to be highly competitive and get the work done on time, within budget.

Features

Realism - Create completely real or totally surreal 3D renders and gaming environments with the addition of 2 new Global Illumination methods, Render To Texture, a highly-interactive physics solution, and even better ways of knowing exactly how your art will look in the final medium well before you hit Render or Export.

Expression - Spend more time adding character to your work, with a new Set-Key Animation System, Spline IK, and easier ways to create and manage animation data, including Draw Curves and a Dope Sheet Editor with Soft Keyframe Selections.

Productivity - Part of the reason why 3ds max software is the most widely-used professional 3D tool in the world is because it allows artists and programmers the freedom to take vastly different approaches to solving the same production problems - that means the flexibility to chose the methods that help you expedite production. 3ds max 5 adds increased connectivity to other 3D software, improved polygon modeling, and delivers dramatically easier methods for manipulating mapping coordinates.

backburner™ is Discreet's new rendering management solution, which gives you 9999 free 3ds max render nodes with each copy of 3ds max software, so you'll be able to manage multiple rendering products, and projects, from the same source.

Innovative Character Management Tools including Bone Tools, a Weight Table for Skin assignments, Spline IK, Progressive Morphing, a Dope Sheet Editor, and vastly improved Function Curve editing tools for easier management of Character Animation elements such as Keyframes and Skinning data.

On-board Radiosity and Advanced Rendering Solutions, such as Toon Shading, Translucency, Area Light Shadows, and Texture Baking, give the artist more choices out of the box, greater quality control, more realistic lighting, and a method for simulating high-resolution scenes on low resolution geometry (awesome for Game Artists).

A re-designed UV unwrap delivers common tools and tasks directly to the artist's fingertips. 3 Unwrapping types (Flatten, Normal, and Unfold) and a slew of selection & navigation tools are now available to make texturing more predictable and easier to manage.

Extended Polygonal modeling tools, with greater degrees of interactivity and better ways of selecting and managing polygons.

A non-destructive SDK which enables 3ds max 4 plug-ins to run without a recompile.

Workflow Enhancements with a new Layer Manager, Improved External References, and Named Selection Set Management for dealing with large amounts of objects.

Easier to learn so you can keep the tools you need right at your fingertips with new enhancements to Quad Menus & the keyboard shortcut system.

Expands with the needs of each project through an extensible architecture that allows studios to add new functionality and mold the 3D environment to their task, need, or preference. Floating network licenses allow easy distribution of 3ds max across your studio.

4.2.3 Maya Unlimited 5.0 (Alias|Wavefront, 2003)

Maya Unlimited 5.0 makes the foremost 3D content creation tools accessible to a broad range of computer graphics professionals in film, broadcast, industrial design, visualization, game development and web design. It is the leading full 3D production solution.

It is also coupled with the industry's most innovative animation and digital effects technology for the creation of advanced digital content.

Features

Intuitive User Interface

Alias|Wavefront™ is a world leader in user interface design. Maya includes an array of ease-of-use tools such as marking menus and 3D manipulators that speed up workflow.

Polygon Modeling

An entire suite of polygon modeling and UV editing tools focused on games, interactive, and general use.

NURBS Modeling

The most advanced curve and surface modeling tools available, based on the award winning NURBS technology by Alias|Wavefront.

Subdivision Surface Modeling

A choice of advanced hierarchical subdivision surface and non-hierarchical polygon mesh tools.

General Animation

A comprehensive range of keyframe and non-linear animation editing tools.

Character Animation

Advanced tools for creating, animating, and editing realistic digital characters.

Deformers

A range of sophisticated deformation tools for modeling and animation.

Rigid and Soft Body Dynamics

High-speed precision manipulation of hard and organic objects determined by physical rules.

Particles and Fields

Fully integrated particle effects controlled by forces based on real-world physics.

Maya Artisan™

Highly intuitive, pressure-sensitive brush interface for digital sculpting and attribute painting.

Maya Paint Effects™

Unique paint technology for easily creating 3D scenes or 2D canvases of unparalleled complexity, detail and realism. Now editable and renderable as polygons!

3D Paint

Paint color, bump, displacement, transparency and other textures directly onto objects.

Multiple Rendering Options

Choose from the Maya native renderer, a new Hardware renderer (graphics card required), mental ray 3.2 and a new Vector renderer capable of producing bitmap and vector outputs including Macromedia Flash®.

Integrated Rendering

A unified rendering workflow provides easy and consistent access to all four renderers through a common user interface.

MEL™

Users can expand Maya's capabilities and add in-house tools via the renowned Maya embedded scripting language. The Maya user interface is fully customizable for animators and technical directors.

Maya API/SDK

A full Application Programmers' Interface opens up Maya's power and functionality for development of plug-ins and translators that extend the range of Maya's functionality or

interface Maya to other systems. The included Maya Software Developers Kit contains extensive plug-in source code examples.

Online Documentation & Tutorials

Comprehensive documentation and tutorials help you hone your Maya skills and take full advantage of what Maya software has to offer. Extensive task-based on-line help and comprehensive reference material search capability provided. A Learning Tool from the extensive library of Maya Learning Tools from Alias|Wavefront is included.

Maya Fluid Effects™

The simulation and rendering of a huge variety of atmospheric, pyrotechnic, viscous liquid, and open water (ocean and pond) effects overcomes one of the greatest barriers in computer animation for all artists.

Maya Cloth™

The production proven and most accurate software solution for simulating a wide variety of digital clothing and other fabric objects.

Maya Fur™

Incredibly realistic styling and rendering of short hair and fur, with Maya Artisan brush interface for painting fur attributes.

Maya Live™ Matchmoving

Precision matching of original live-action footage with 3D elements rendered in Maya.

4.2.4 SOFTIMAGE|3D (Softimage, 2003)

SOFTIMAGE®|3D is Softimage's legendary 3-D character animation product for the film, commercial/broadcast and games development markets. SOFTIMAGE|3D features robust, production proven organic modeling, legendary character animation tools and high-quality photorealistic rendering—providing a perfect first step into the world of 3-D production. New version 4.0 offers a range of new features with an emphasis on game authoring including multiple UV texturing, vertex color authoring and polygon hide/unhide tools. SOFTIMAGE|3D also offers an easy upgrade path to the next-generation SOFTIMAGE|XSI™ nonlinear animation (NLA) system.

Features

Interactive Games Tools

SOFTIMAGE|3D games development environment provides a selection of platform-specific tools and exporters for on-target platforms like Sony PlayStation, PC/DirectX and Nintendo 64. Take maximum advantage of platform and rendering options with unbeatable on-target viewing tools and target-specific rendering-attribute editors. SOFTIMAGE|3D also features import and export of the Softimage dotXSI™ v.3.0 file format, designed especially for interactive media applications. The dotXSI file format allows ASCII import and export of characters, models, and animation for complete customizability of any game development pipeline. SOFTIMAGE|3D offers vertex color manipulation and authoring, including alpha-channel support, as well as powerful UV texture editing, and texture pre-lighting with Rendermap** to capture sophisticated mental ray lighting and effects directly in texture maps.

Intuitive Workflow

SOFTIMAGE|3D is well known for its intuitive, animation-oriented workflow. Tools are specifically designed for integration into the overall production pipeline, providing rapid, high-quality results to meet the most demanding deadlines. The tools are also where an artist expects them to be, allowing the creative process to flow, so artists can focus on their creations.

Open Extensible Environment

SOFTIMAGE|3D is a continually evolving system, refining existing tools and providing powerful customization avenues, both within the product and through the SOFTIMAGE|SDK (Software Development Kit), SOFTIMAGE|GDK (game development kit), and XSI Viewer Tools*.

4.2.5 Adobe Photoshop 7.0 (Adobe, 2003)

Photoshop provides a comprehensive toolset, unmatched precision, and powerful creative options to help to create professional-quality images for Web, print, and emerging media such as wireless devices. And when using Photoshop in tandem with other Adobe software, the advantage of superior Adobe technologies such as crossproduct color management tools, Smart Object technology, and transparency can be taken.

Features

Work more efficiently

From file management to workspace controls to editing multiple steps at one time. Photoshop gives you the tools you need to keep the work on track and bring it in on deadline.

Edit images with ease

Photoshop delivers high-powered image editing, photo retouching, and compositing tools to help you get professional-quality results.

Enjoy unlimited creative options

With innovative special-effect options and powerful painting and drawing tools, there's no limit to the results you can achieve with Photoshop.

Create compelling Web designs

Produce exceptional imagery for the Web and wireless devices with Photoshop and ImageReady, which ships with Photoshop.

Enjoy precise typographic control

Photoshop delivers professional-quality type controls to help you create imagery that communicates with precision and style.

Automate repetitive tasks

Streamline and simplify the production process by turning time-consuming jobs into automated operations.

Develop a reliable workflow

Keep files moving efficiently from the beginning of the process to the end.

Maintain color precisely

Keep color consistent across different devices and count on reliable output to any media.

4.2.6 Macromedia Fireworks MX (Macromedia, 2003)

Macromedia Fireworks MX has the familiar tools that graphics professionals demand, brought together in a single, web-centered environment. Quickly create original web graphics and interactivity, from simple graphical buttons to complex rollover effects and pop-up menus. Easily edit and seamlessly integrate source files in all the major graphics formats, and export to Macromedia Flash and Dreamweaver projects. Fireworks MX delivers a complete graphics toolset with a workflow that promotes teamwork and enhances productivity.

Features

Enhance productivity with a highly customizable workspace shared between Macromedia Flash MX, Dreamweaver MX, and Fireworks MX. The Property Inspector provides a single panel solution to edit object, text, and tool properties.

Quickly create sophisticated web navigation. Wizards automatically generate the graphics and JavaScript. Easily use and edit the resulting files in Dreamweaver MX.

Automatically generate data-driven graphics. By dynamically linking to XML content, you can automate repetitive graphics creation. Fireworks creates individual graphics based on each dataset. To create graphics from existing sites, point to an HTML file and automatically reassemble the images, text rollovers, and pop-up menus into a new source file.

Get professional design results. Seamlessly edit vectors and bitmaps in one integrated environment that automatically responds to selections. Robust bitmap-editing support works exactly the way you expect for creating new bitmaps and fine-tuning images.

Optimize files and streamline exports to industry-leading applications. With a single click, Fireworks MX automatically makes intelligent export choices to Macromedia Flash, Dreamweaver, FreeHand, and Director Shockwave Studio, and to third-party graphics and HTML packages.

Extend functionality and automate tasks by creating commands that combine a powerful JavaScript extensibility API with interfaces developed in Macromedia Flash MX.

Effortlessly share files across graphics applications including Macromedia Flash and FreeHand, and Adobe Photoshop and Illustrator. Export SWF files and open Fireworks files directly within Macromedia Flash MX. Open, edit, and export Photoshop graphics while retaining layers, masks, and text properties.

Integrate seamlessly with leading editors including the Roundtrip Editor in Macromedia Dreamweaver MX and Microsoft FrontPage. Round-trip graphics and HTML between Fireworks and Dreamweaver MX or FrontPage with a single click in the HTML editing environment.

Develop for the latest standards using support for web accessibility and XHTML. Check for compliance with Section 508 requirements and export compliant HTML. Round-trip XHTML with Macromedia Dreamweaver MX.

4.3 Development Tools Chosen

Microsoft Visual C++ .Net 2003

• To create the Virtual Reality Training tool application program

3ds max™ 5

• To create 3D objects

Macromedia Fireworks MX

• To create and edit images

4.4 System Requirements

4.4.1 Hardware

- Pentium III 1.2 GHz (or faster processor)
- 512 MB SDRAM
- 20 GB Hard Disk space (or larger)
- GeForce 2 Graphics Card (or higher)
- Floppy Drive
- CD ROM Drive
- Monitor
- Keyboard
- Mouse

4.4.2 Software

- Windows 98/2000/XP
- Microsoft Visual C++ .Net 2003
- 3ds max™ 5
- Macromedia Fireworks MX

CHAPTER 5 SYSTEM DESIGN

5.1 OVERVIEW

5.2 SYSTEM FUNCTIONALITY DESIGN

5.3 USER INTERFACE DESIGN

5.0 System Design

5.1 Overview

System design is a phase of the Waterfall model where system's requirements are translated into system characteristics. System requirements had been discussed in the previous chapter. The objectives of system design are listed below.

- Specify logical design elements

  Plan a detail design specification with specific logical elements that describe the features of a system.

- Meet user's requirements

  Incorporate user's needs into the system in terms of appropriate procedure and performance, accurate results and overall reliability.

System design consists of

i. System Functionality Design

ii. User Interface Design

5.2 System Functionality Design

System Functionality Design is presented in the form of a hierarchy chart. It identifies the major modules in the system. In addition, high level components can also be broken down into sub-modules.

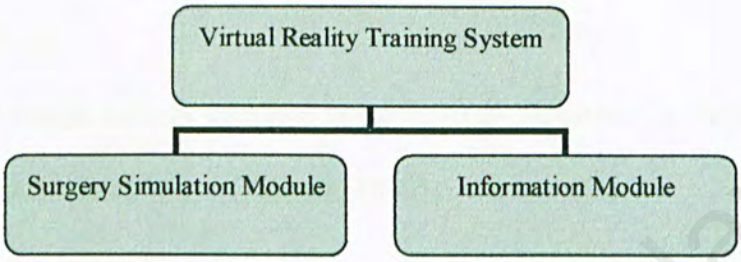

Figure 5.1 Virtual Reality Training System Modules

5.3 User Interface Design

Interface Design is the specification of a conversation between system users and the computer. A good, easy to use and user friendly interface will make user's job easier and more pleasant (Hawryszkiewyez, 1998).

5.3.1 Screen Design

A good screen design reduces interface complexity as perceived by the user. The design will follow the guidelines below (Fertuck, 1997).

Concreteness

It is easier to work with concrete objects than with an abstract concept. Icons that perform specific functions will be incorporated into the system to achieve this purpose.

Visibility

It is easier to operate an interface with the commands rather than remember options from the language syntax. This is due to proven scientific research that one has highly developed powers of pattern recognition but relatively poor memory. Therefore, commands, function keys, options and help will be visible on the screen.

Simplicity

One of the best guideline for screen design is KISS- Keep It Simple Simon! Design element will be used judiciously.

Context Sensitivity

Options and commands that are available depend on the current state of the screen. Commands that are inappropriate in the current mode should not be made available.

CHAPTER 6 SYSTEM DEVELOPMENT

6.1 DEVELOPMENT ENVIRONMENT

6.2 HARDWARE REQUIREMENTS

6.3 SOFTWARE REQUIREMENTS

6.4 SYSTEM'S MODULES

6.0 System Development

6.1 Development Environment

A system is affected by the development environment. Suitable hardware and software is pivotal in a development process, as it fastens the development process and increases the system's reliability.

6.2 Hardware Requirements

- Pentium III 1.2 GHz (or faster processor)
- 512 MB SDRAM
- 20 GB Hard Disk space (or larger)
- GeForce 2 Graphics Card (or higher)
- Floppy Drive
- CD ROM Drive
- Monitor
- Keyboard
- Mouse

6.3 Software Requirements

Software	*Type*	*Notes*
Windows 98/2000/XP	Operating System	System's Platform
Microsoft Visual C++ .Net 2003	System Development	Coding
3ds max™ 5	System Development	3d Modeling
Macromedia Fireworks MX	System Development	Image Modification
Microsoft Word	Documentation	Report Writing

Table 6.1: Software Requirements

6.4 System's Modules

There are 5 source files and 5 header files for this system.

Source Files

- Main.cpp
- gl3ds.cpp
- glArcBall.cpp
- glImage.cpp
- glFont.cpp

Header Files

- Main.h
- gl3ds.h
- glArcBall.h
- glImage.h
- glFont.h

Main.cpp	
Function	***Description***
ChangeScreenResolution	Changes The Screen Resolution
Cone	Draw a cone
CreateTexture	Create texture
CreateWindowGL	Creates the OpenGL Window
Cystic	Draws the Cystic Dust
Deinitialize	User Deinitialization
DestroyWindowGL	Destroy The OpenGL Window and Release Resources
Draw	Draws the OpenGL Scene
DrawGLInfo	Draws the OpenGL Information
InitGL	Setup For OpenGL
Initialize	GL Init Code and User Initialiazation
KeyMovement	Keypad movement
LoadGLTextures	Load texture
LoadTexture	Load texture
ReSizeGLScene	Resize And Initialize The GL Window
RegisterWindowClass	Register A Window Class For This Application
ReshapeGL	Reshape The Window When It's Moved Or Resized
TerminateApplication	Terminate The Application
ToggleFullscreen	Toggle Full screen /Windowed
Update	Perform Motion Updates Here
WinMain	Program Entry (WinMain)
WindowProc	Setup Windows

bSetupPixelFormat	Setup the Pixel Format

Table 6.2: Main.cpp Functions and Descriptions

gl3ds.cpp	
Function	***Description***
CLoad3DS	This constructor initializes the tChunk data
CleanUp	Cleans up our allocated memory and closes the file
ComputeNormals	Computes the normals and vertex normals of the objects
GetString	Reads in a string of characters
Import3DS	Called by the client to open the .3ds file, read it, then clean up
ProcessNextChunk	Reads the main sections of the .3DS file, then dives deeper with recursion
ProcessNextMaterialChunk	Handles all the information about the material (Texture)
ProcessNextObjectChunk	Handles all the information about the objects in the file
ReadChunk	Reads in a chunk ID and it's length in bytes
ReadColorChunk	Reads in the RGB color data
ReadObjectMaterial	Reads in the material name assigned to the object and sets the materialID
ReadUVCoordinates	Reads in the UV coordinates for the object
ReadVertexIndices	Reads in the indices for the vertex array
ReadVertices	Reads in the vertices for the object

Table 6.3: gl3ds.cpp Functions and Descriptions

glArcBall.cpp	
Function	***Description***
ArcBall_t	Constructor / Destructor

Table 6.4: glArcBall.cpp Functions and Descriptions

glImage.cpp	
Function	***Description***
ImageLoad	Load image
Getint	Get integer
Getshort	Get short

Table 6.5: glImage.cpp Functions and Descriptions

glFont.cpp	
Function	***Description***
BuildFont	Build the Font
GetListBase	Get the List Base
GetTexture	Get the Texture
SetFontTexture	Set the Font Texture
SetWindowSize	Set the Window Size
glFont	Constructor
glPrintf	Prints out the String
~glFont	Destructor

Table 6.6: glFont.cpp Functions and Descriptions

glArcBall.h	
Function	***Description***
Vector3fCross	Sets this vector to be the vector cross product of vectors v1 and v2
Vector3fDot	Computes the dot product of this vector and vector v1.
Vector3fLength	Returns the length of this vector
Vector3fLengthSquared	Returns the squared length of this vector

Table 6.7: glArcBall.h Functions and Descriptions

CHAPTER 7 SYSTEM TESTING

7.0 System Testing

7.1 Overview

System errors and failures occur mainly because of inadequate or improper testing. The purpose of testing is to detect the presence of errors in system- the ones that have not been discovered yet.

7.2 Types of Faults

Fault Class	*Static Analysis Check*
Data Faults	▪ Variables used before initialization ▪ Variables declared but never used ▪ Variables assigned twice but never used between assignments ▪ Possible array bound violations ▪ Undeclared variables
Control Faults	▪ Unreachable code ▪ Unconditional branches into loops
Input/Output Faults	▪ Variable output twice with no intervening assignment
Interface Faults	▪ Parameter type mismatches ▪ Parameter number mismatches ▪ Non-usage of the results of functions ▪ Uncalled functions and procedures
Storage Management Faults	▪ Unassigned pointers ▪ Pointer arithmetic

Table 7.1: Static Analysis Check

Fault Class	*Inspection Check*
Data Faults	▪ Are all program variables initialized before their values are used? ▪ Have all constants been named? ▪ Should the upper bound of arrays be equal to the size of the array or size -1? ▪ If character strings are used, is a delimiter explicitly assigned? ▪ Is there any possibility of buffer overflow?
Control Faults	▪ For each conditional statement, is the condition correct? ▪ Is each loop certain to terminate? ▪ Are compound statements correctly bracketed? ▪ In case statements, are all possible cases accounted for? ▪ If a break is required after each case in case statements, has it been included?
Input/Output Faults	▪ Are all input variables used? ▪ Are all output variables assigned a value before they are output? ▪ Can unexpected inputs cause corruption?
Interface Faults	▪ Do all function and method calls have the correct number of parameters? ▪ Do formal and actual parameter types match? ▪ Are the parameters in the right order? ▪ If components access shared memory, do they have the same model of the shared memory structure?
Storage Management Faults	▪ If a linked structure is modified, have all links been correctly reassigned? ▪ If dynamic storage is used, has space been allocated correctly? ▪ Is space explicitly de-allocated after it is no longer required?
Exception Management Faults	▪ Have all possible error conditions been taken into account?

Table 7.2: Inspection Check

7.3 Testing Principles

Two common testing principles were incorporated into the system testing phase. They are the structural and functional testing.

Structural (usually called "white box") testing, and functional ("black box") testing have unique characteristics, advantages and limitations that make them more or less applicable to certain stages of test.

7.3.1 White Box Testing

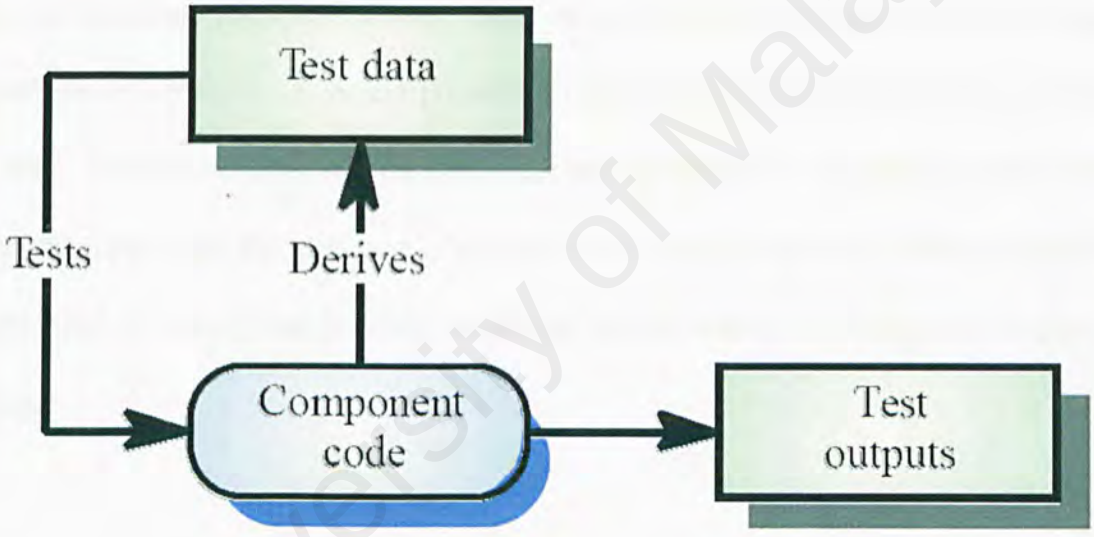

Figure 7.3: White Box Testing

Structural tests verify the structure of the software itself and require complete access to the object's source code. This is known as 'white box' testing because one could see into the internal workings of the code.

White-box tests make sure that the software structure itself contributes to proper and efficient program execution. Complicated loop structures, common data areas, 100,000

lines of code and nests of ifs are recipes to system crashes. The system should have well-designed control structures, sub-routines and reusable modular programs.

Many studies show that the single most effective defect reduction process is the classic structural test - the code inspection or walk-through. Code inspection is like proofreading - it can find the mistakes the author missed - the "typo's" and logic errors that even the best programmers can produce. Debuggers are typical white-box tools.

White-box testing's strength is also its weakness. The code needs to be examined – by highly skilled technicians. That means that tools and skills are highly specialized to the particular language and environment. Also, large or distributed system execution goes beyond one program, so a correct procedure might call another program that provides bad data. In large systems, it is the execution path as defined by the program calls, their input and output and the structure of common files that is important. This gets into a hybrid kind of testing that is often employed in intermediate or integration stages of testing.

7.3.2 Black Box Testing

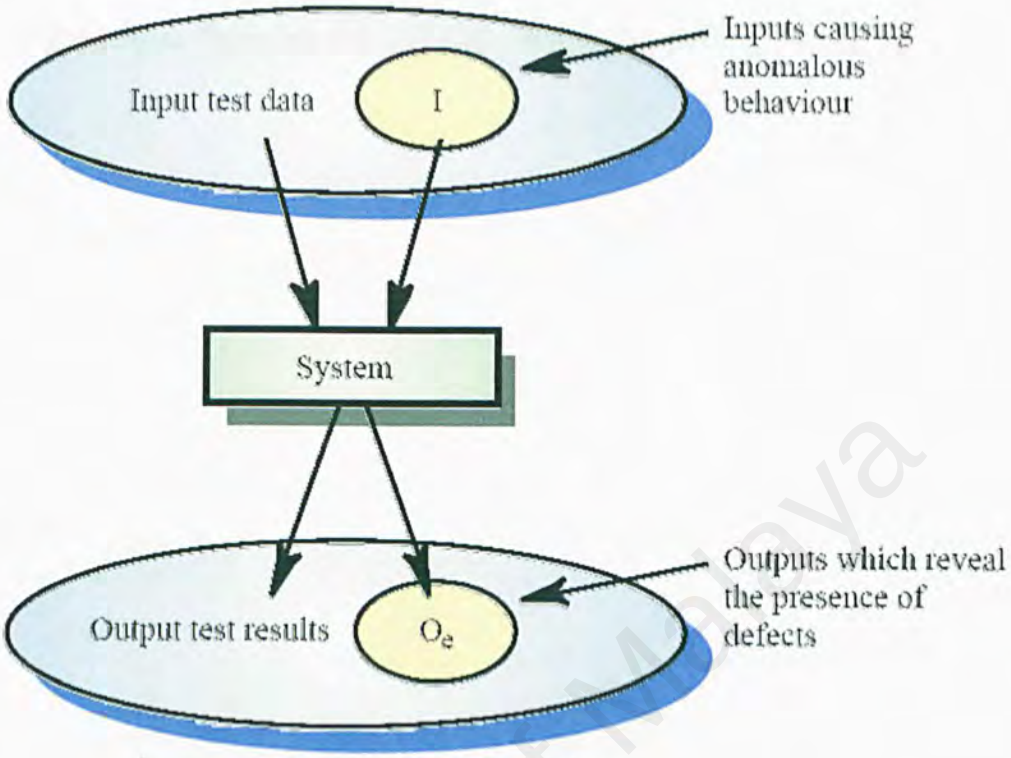

Figure 7.4: Black Box Testing

Functional tests examine the observable behavior of software as evidenced by its outputs without reference to internal functions. Hence the term 'black box' testing. If the program consistently provides the desired features with acceptable performance, then specific source code features are irrelevant. It's a pragmatic and down-to-earth assessment of software.

Black box tests better address the modern programming paradigm. As object-oriented programming, automatic code generation and code re-use becomes more prevalent, analysis of source code itself becomes less important and functional tests become more important.

Black box tests are also use to achieve a certain quality target. Since only the people paying for an application can determine if it meets their needs, it is an advantage to create the quality criteria from this point of view from the beginning.

7.4 Testing Organization

System testing is divided into several stages.

7.4.1 Unit Testing

In some organizations, a peer review panel performs the design and/or code inspections. Unit or component tests usually involve some combination of structural and functional tests by programmers in their own systems. Component tests often require building some kind of supporting framework that allows components to execute.

7.4.2 Integration Testing

The individual components are combined with other components to make sure that necessary communications, links and data sharing occur properly. It is not truly system testing because the components are not implemented in the operating environment. The integration phase requires more planning and some reasonable sub-set of production-type data. Larger systems often require several integration steps.

There are three basic integration test methods:

- all-at-once

  The all-at-once method provides a useful solution for simple integration problems, involving a small program possibly using a few previously tested modules.

- bottom-up

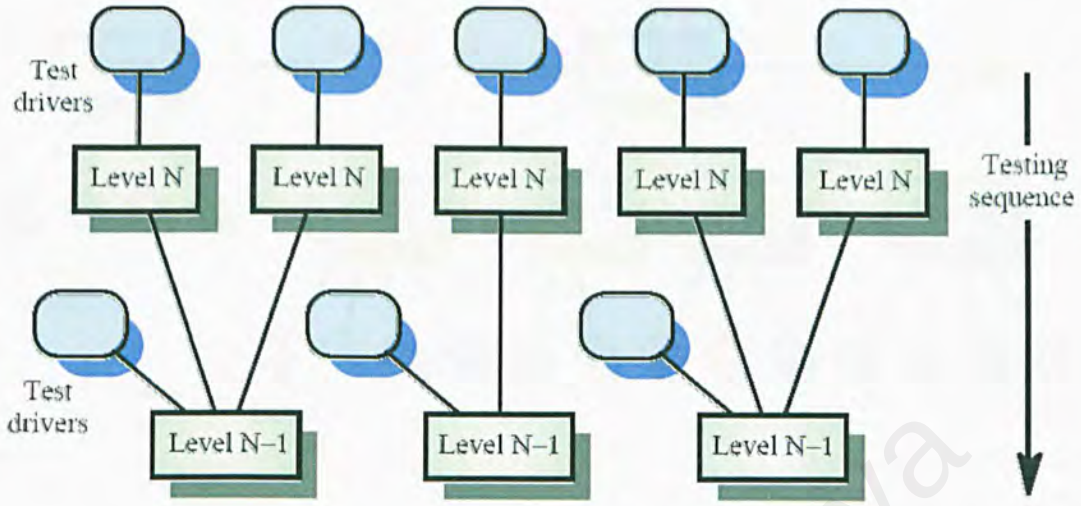

Figure 7.5: Bottom - Up Integration Testing

Bottom-up testing involves individual testing of each module using a driver routine that calls the module and provides it with needed resources. Bottom-up testing often works well in a less structured environment because there is less dependency on availability of other resources to accomplish the test. It is a more intuitive approach to testing that usually finds errors in critical routines earlier than the top-down method. However, in a new system, many modules must be integrated to produce system-level behavior, thus interface errors surface late in the process.

- top-down

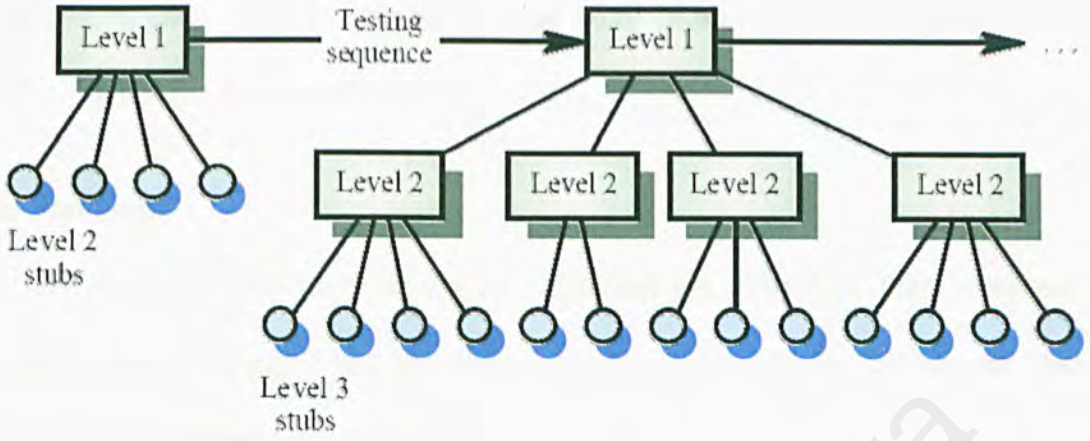

Figure 7.6: Top - Down Integration Testing

Top-down testing fits a prototyping environment that establishes an initial skeleton that fills individual modules into it. The method lends itself to more structured organizations that plan out the entire test process. Although interface errors are found earlier, errors in critical low-level modules are found later.

What all this implies is that a combination of low-level bottom-up testing works best for critical modules, while high-level top-down modules provide an early working program that can give management and users more confidence in results early on in the process. There may be need for more than one set of integration environments to support this hybrid approach.

7.4.3 System Testing

The system test phase begins once modules are integrated enough to perform tests in a whole system environment. System testing can occur in parallel with integration test, especially with the top-down method.

7.5 Test Planning

Careful test planning helps us to design an organized test. The Test Plan takes into account the system's objective and incorporates the project deadlines. We use the Test Plan as a guide to organize testing activities.

Test Plan for VR Training: Laparoscopic Surgery Open Framework

Function	Test Cases	Expected Result	Actual Results	Remarks
1. Wire frame mode	Toggle wire frame mode by pressing the key 'F2'	The 3d models goes into wire frame mode		
2. Lighting	Toggle lighting by pressing the key 'F3'	Lighting are turned on		
3. Selects Hook Electrode (Left)	Selects Hook Electrode (Left) by pressing the key 'F4'	Hook Electrode (Left) are selected		
4. Selects Hook Electrode (Right)	Selects Hook Electrode (Right) by pressing the key 'F5'	Hook Electrode (Right) are selected		
5. Selects Retract	Selects Retract by pressing the key 'F6'	Retract are selected		
6. Grasps using Retract	Grasps using Retract by pressing the key 'F7'	The gallbladder are grasped using the retract		
7. Gallbladder removal	Gallbladder removed using the retract by holding the key 'F7' and 'A'	The gallbladder removed using the retract		

8. Selects Straight Scissors	Selects Straight Scissors by pressing the key 'F8'	Straight Scissors are selected		
9. Cutting using Straight Scissors	Cut the arteries using the straight scissors by letting go the key 'F8'	The simulation of scissors cutting		
10. Turns on Information and Lighting	Turns on information and lighting by pressing the key '1'	Information and Lighting are turned on		
11. Turns off Information and Lighting	Turns off information and lighting by pressing the key '2'	Information and Lighting are turned off		
12. Exits the Program	Exits the Program by pressing the key 'ESC'	The program closes safely		

Table 7.7: The System's Test Plan

7.6 Maintenance

Software maintenance is the general process of changing a system after it has been delivered. The changes may be simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancements to correct specification errors or accommodate new requirements (Sommerville, 2001).

There are two different types of software maintenance that is incorporated into the system testing phase:

- Maintenance to repair software faults

  Coding errors are usually relatively cheap to correct; design errors are more expensive as they may involve the rewriting of several program components. Requirements errors are the most expensive to repair because of the extensive system redesign which may be necessary.

- Maintenance to add to or modify the system's functionality

  This type of maintenance is necessary when the system requirements change in response to organizational or business change. The scale of the changes required to the software is often much greater than for the other types of maintenance.

CHAPTER 8 SYSTEM REVIEW

8.1 PROBLEMS AND SOLUTIONS

8.2 ADVANTAGES OF VIRTUAL REALITY TRAINING

8.3 SYSTEM CONSTRAINTS

8.4 FUTURE ENHANCEMENTS

8.5 CONCLUSION

8.0 System Review

8.1 Problems and Solutions

- *Limited Development Time*

To develop a fully functional Virtual Reality Training tool in just two months time is near to impossible. There was too much coding and analysis that needs to be done.

Solutions:

Time management is critical to ensure that the project schedule is followed.

- *Lack of Knowledge in Authoring Tools*

I lack the experience and knowledge in dealing with the authoring tools. Familiarity with the authoring tools functions were also a problem during the system development phase.

Solutions:

A lot of researches were done which includes referring to online tutorials and reference books that are available on the Internet and library.

8.2 Advantages of Virtual Reality Training

The Virtual Reality Training: Laparoscopic Cholecystectomy Surgery Open Framework provides a safe, controllable environment for users to learn, allowing them to make mistakes without consequences to the patient.

8.3 System Constraints

- The laparoscopic cholecystectomy surgery simulation includes only basic instrument control.
- Due to the expensive and unavailability of hardware such as robotic arms, simulation controls will be constraint to only keyboard and mouse.

8.4 Future Enhancements

There are several future enhancements that can be done to improve the system which includes:

- Collision Detection

  The system should be able to detect collision between the simulation scene and the camera and selected surgery tools.
- Puncturing of the cystic duct and the simulation of blood splatter

  To incorporate an algorithm to pinpoint the location where the hook electrode punctures the cystic duct and at the same time simulate blood splatter.
- Robotic Arms Usage

  To incorporate robotic arms into the system as inputs instead of conventional mouse and keyboard.

8.5 Conclusion

Upon the completion of this project, I have learnt to face challenges with an open mind and attempt to perform my best in everything. The knowledge and experience that I have gained were invaluable.

Having been through the development of a real life project in a real working environment, I can better prepare myself for the future. The mistakes that I have made and the experience that came while looking for solutions had made me realized that for every problem there will always be solutions to it. By constantly doing research, it will not only increase our knowledge but also enhance our mind towards thinking creatively.

I am confident that the experience that I had gained will provide me with a strong foundation towards my career and achievements in the near future.

Reference

Adobe. [Online]. Available:
<http://www.adobe.com/products/photoshop/pdfs/overview.pdf>
[2003, June 23].

Alias|Wavefront. [Online]. Available:
<http://www.alias.com/eng/products-services/maya/file/maya5_specsheet.pdf>
[2003, June 23].

Discreet. [Online]. Available:
<http://www.discreet.com/docs/products/3dsmax/3dsmax5_techspec.pdf>
[2003, June 23].

Fertuck, Len. (1997). System Analysis and Design with Modern Method.
Business and Educational Technologies.

GridSet Exchange. [Online]. Available:
<http://gridset.com/flyers/factsheet3.pdf> [2003, August 26].

Hawryszkiewyez, I. et al. (1998). Introduction to System Analysis and Design.
Prentice Hall.

Institute for Applied Informatics (IAI) of the Forschungszentrum Karlsruhe. [Online].
Available: <http://wwwserv2.iai.fzk.de/~artemis/welcome_engl.html>
[2003, August 26].

Kilgard, Mark J. (1996, November, 13). The OpenGL Utility Toolkit (GLUT)
Programming Interface API Version 3. Silicon Graphics, Inc.

Macromedia. [Online]. Available:
<http://www.macromedia.com/software/fireworks/productinfo/overview/firewor
ks_datasheet.html> [2003, June 23].

Medina, Marelyn. Laparoscopic Surgery. [Online]. Available:
<http://www.sls.org/patientinfo/aboutlap.html> [2003, June 25].

Microsoft Corporation. [Online]. Available:
<http://msdn.microsoft.com/visualc/productinfo/features/default.aspx>
[2003, June 23].

Rapid Application Development. [Online]. Available:
<http://csweb.cs.bgsu.edu/maner/domains/RAD.htm> [2003,June 16].

Simbionix. [Online]. Available:
<http://www.simbionix.com/LAP_Mentor.html> [2003, August 26].

Softimage. [Online]. Available:
<http://www.softimage.com/Products/3d/v4/Datasheet/datasheet_3D_v.4.0.pdf>
[2003, June 23].

Sommerville, Ian. (2001) Software Engineering
(6th Edition). Canada: Addison-Wesley Publishing Company.

Spiral Lifecycle. [Online]. Available:
<http://cctr.umkc.edu/~kennethjuwng/spiral.htm> [2003, June 16].

The KISMET 3D-Simulation Software. [Online]. Available:
< http://iregt1.iai.fzk.de/> [2003, August 26].

Upton, Graham. Laparoscopic Surgery Simulation Realism In A PC. [Online].
Available:
<http://www.dvc400.com/papers/LapSim.pdf> [2003, June 26].

VEST System *One* (VS*One*) Technology. [Online]. Available:
<http://iregt1.iai.fzk.de/KISMET/VestTech.html> [2003, August 26].

Virtual Endoscopic Surgery Training. [Online]. Available:
<http://iregt1.iai.fzk.de/KISMET/VestSystem.html> [2003, August 26].

Woo, Mason. et al. (1997). The Official Guide to Learning OpenGL, Version 1.1
(2nd ed.). Canada: Addison-Wesley Publishing Company.

Your Medical Resource. [Online]. Available:
<http://www.yourmedicalsource.com/library/laparoscopy/LAP_whatis.html>
[2003, June 25].