HLA BASED SIMULATOR OF A GROUND COMBATTING SCENARIO USING FIGHTING OBJECTS

By

NC Ammar Sohail (Leader)

PC Abdur Rehman

Capt Qaisar Abbas

Capt Irfan Gul

Submitted to the Faculty of Computer Science Military College of Signals
National University of Sciences and Technology, Rawalpindi In Partial Fulfillment For
The Requirements of a B.E. Degree In Computer Software Engineering
Apr 2006

Abstract

War scenarios existed from earlier centuries. Now weapons are modernized and strategies are well planned before any operation of war. Since many elements and phases of war are continuous, they become candidates of simulation. Simulating war scenes helps planning for real time encounters. Simulation can incorporate element such as machine gun to huge elements such as nuclear instalments and tests. We can even simulate possible actions and strategies of enemy and own forces.

Training plays important role in military life and a huge amount is spent on realistic training of troops. However, the training involves heavy running expense even in the peace time, and can cost heavily in terms of resources. At times we have the shortage of training Equipment or if at all it is available then there is too much wear and tear by its extensive use. As far as ground combat is concerned, till now Pakistan Army does not have any true Working Simulator. The simulator to be used must be very realistic in terms of visual accuracy, level of detail. It will be consisting of various mobile fighting objects (tanks, weapon heads, soldiers etc).

As a starting point, one specific scenario of battle field is depicted which includes the elements of both sides (Attacker and Defender). Terrain being the basic and primary concern is being built as a separate module. Tanks along with their mechanics are another module. Whereas few other elements of battle field are also incorporated. A ground combat scenario will be simulated with certain limitations.

This application will run on a distributed network. We have chosen HLA architecture for its implementation, that supports huge complex simulations.

Declaration

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institute or elsewhere.

Dedication

We would like to dedicate this project to our teachers for their unwavering support and guidance and to the people who still want to contribute something to this country.

Acknowledgements

We gratefully acknowledge the continuous guidance and motivation provided to us by our project advisors Dr. Shoaib Ahmed(EME) and Asst. Prof. Tauseef Rana(MCS). Without their personal supervision, advice and help, timely completion of this project would have been impossible. Our very special thanks are extended to Head of Department Col. Raja Iqbal for his ever extended moral and technical support. We would also like to thank our esteemed instructors who helped us to nurture our capabilities, giving us the skills and the confidence to take on challenges and emerge victorious.

We are also deeply indebted to our families for their never ending patience and support for our mental peace and to our parents for the strength that they gave us through their prayers.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Armr Sqn	Armour squadron
Attk	Attack
AMG	Architecture Management Group
API	Application Programming Interface
BMP	Bit Map
Bkwd	Backward
Bn Sp	Battalion Support
CO	Commanding Officer
Coy	Company
Ctr	Counter
DLL	Dynamic Link Library
DOF	Degree of Freedom
Def	Defence(s)
DF	Defensive Fire
DCB	Ditch Cum Bund
Fwd	Forward
FPS	Frames per second
FU	Firing Unit
FOV	Field of View
FED	Federation
FOM	Federation Object Model
Gr Cbt	Ground Combat
GUI	Graphical User Interface
HLA	High Level Architecture
HTTP	Hyper Text Transport Protocol
HTML	Hyper Text Markup Language
JPEG	Joint Photographic/Picture Expert Group
LCD	Liquid Crystal Display

MOM	Management Object model
Posn	Position
Rd	Road
RGB	Red, Green,Blue
RTI	Runtime Infrastructure
Recce	Reconnaissance
SOM	Simulation Object Model
Tk	Tank
Tgts	Target
US DOD	US Department of defence

Chapter 1

1. Introduction and background

1.1 Introduction

Today, one of the most powerful outcomes of the digital computer has been its role in representing real physical environments through modelling and simulation. A “model” is a representation of a real physical system. Often, physical principles are the cornerstone for creating mathematical models, be it ordinary differential equations, distributed parameter systems (partial differential equations), heuristic models based on system identification or machine learning. Simulation utilizes a model (mathematical or otherwise) to make a hypothetical realization of a real physical system. Simulation allows open-loop or closed-loop systems analysis and provides the user with numerous alternatives based on “what if" situations to be realized. In many circumstances, several physical models and simulations need to take place simultaneously. This requirement leads to the notion of distributed modelling and simulation.

1.2 Overview/aim

The aim of undertaking this project derives its roots from the need to train the troops and officers of Pakistan Army in operations of war. The aim is to provide necessary tools through simulation for the training of personnel in special operations of war without going into the exercise area. The project objectives include creation of a general purpose system in the frame work of the specific instance of one of the most important operation of war i.e. ‘Defense’ using the HLA architecture. Hence a simulator is required to replace the actual training operations. This project will enhance the capabilities of mission/training oriented organizations like SI & T (School of Infantry and Tactics), Staff College and other Army institutions of our country as it is feasible and cost effective for these institutions.

Along with Simulation Developers, Virtual environment researchers, science-fiction authors, and video game players share a common dream: the existence of an always-on, globally connected, infinitely expandable network of virtual worlds. Ideally, this network would subsume all types of virtual environments currently available without inheriting their limitations. The worlds of such a network could be used for social interaction, but they would share neither the superficial level of interactivity nor the impermanence of three dimensional chat worlds such as Adobe Atmosphere. They could be used for gaming, but they would not be subject to the limited scalability of first-person shooting games such as “Delta Force 03”, nor to the monopolistic control exercised by the hosts of massively multiplayer online role-playing games such as Ever Quest. They would form the perfect setting for virtual environment research, for they would be as robust and enduring as current research environments are brittle and short-lived.

To those who would attempt it, the establishment of such a network presents an intimidating challenge: the creation of a set of standards that would formalize certain aspects of virtual environment applications without limiting their overall functionality. These standards would have to be general enough to apply to all existing types of environments, flexible enough to accommodate new types of environments, evolvable enough to allow for extension and improvement over time, and simple and well-specified enough to encourage widespread adoption. The standards on which the World Wide Web is based—the Hyper-Text Transport Protocol (HTTP) and the Hyper-Text Mark-up Language (HTML)—provide an excellent example of how to achieve the kind of universality and flexibility required. As with those associated with the World Wide Web, a generalized set of standards for virtual environments is likely to include at least two categories: formats for static data representation, like HTML, and protocols for dynamic data interchange, like HTTP. Here we are concerned with the development of a standard architecture of the second category: a mechanism for transmitting dynamic state information between virtual environment applications.

1.3 Background

The development of Interoperable simulations has been a major concern of the developed countries for the past few years. The American Department of Defense has been in particular been involved in large scale complex simulations that have the capability of integrating multiple simulations and Human in the loop. After attempts like the Development of DIS and ALSP that did not provided Universal Interoperability, the US DOD launched a Major project in collaboration with Miter Corporation and the OMG, to develop a standard for interoperability.

The HLA was developed based on a process involving government, academia, and industry. In 1995, three industry teams developed concepts for the definition of a high level architecture. The results of these efforts were combined along with additional insights from other modeling and simulation projects to arrive at the initial definition of the HLA. On 31 March 1995, this definition was presented to an Architecture

Management Group (AMG) formed to develop the HLA from this initial definition. The AMG developed the architecture based on cooperative efforts to apply the HLA in a series of prototypes designed to ensure the HLA addresses the broad set of application requirements. The result was a baseline HLA definition, completed in August 1996. The AMG has continued to evolve the HLA based on experience with its use, reviewing and updating the HLA specifications on a six-month cycle. In February 1998, version 1.3 of the HLA specification was adopted (Defense Modeling and Simulation Office 1998a) (Defense Modeling and Simulation Office 1998b) (Defense Modeling and Simulation Office 1998c). The HLA is already a standard for use in the US Department of Defense (U.S. Department of Defense USD(A&T) 1996), in NATO (North Atlantic Treaty Organization 2000), in the Object Management Group (OMG 2001 as Distribute Simulation Systems) and has also been formally approved by the IEEE as HLA-1516-2000. [1]

The first prototype of an HLA that supports interoperability was developed by the US DOD; other attempts are still under way at Research and Educational institutes. Some private companies are also engaged in development of HLA compliant products, but there sale is restricted by special export restrictions from US DOD.

1.4 Problem to be solved

Now a days Simulation has become one of the most widely used techniques in operations research and management sciences. It is No longer the approach of “last resort”. According to Estimates 75% of total computing power in the World is used for various kinds of simulations. But the question arises why we need the simulations that consume so many resources? The requirement of such processing powers can be anticipated after a thorough search. Simulation tools are used to model complex systems to describe the behavior of the system in different circumstances, or even to predict their future responses. These scenarios are sometimes too dangerous, expensive or difficult to practice for real, e.g. collision testing of automobiles. Simulations are used for Training and Educational purposes, like Commercial and military pilots utilize interactive simulations to enhance their flying skills. They are also used for Analyzing processes which in real life have a time spans of days / years or eons. For example: Astronomers may analyze theories that might otherwise take millions of years to verify. And most importantly sometimes the system to be simulated is so complex, that its interactions can be treated only through simulations. (They are Not Possible to be carried out in real life) For example: Seismologists use simulations to predict the effects of earthquakes.

In many of these circumstances, several physical models and simulations need to take place simultaneously. This requirement leads to the notion of distributed modeling and simulation. In order to appreciate distributed modeling and simulation, let us use as an example the problem on hand in NASA. Assume that we are to simulate numerous objects or agents, e.g. rovers like the Mars Pathfinder, on earth-like or alien terrain. These rovers are to survey the planet and make observations of their surroundings and report to a central location, on Earth or in space. In such scenarios one can have many physics-based models and many simulations of various types running at the same time. Some of these models can be the rovers, blowing wind, rocks rolling down a steep slope, a satellite etc. All of these physical models need to be simulated simultaneously. Simulating such a scenario without a distributed approach is extremely difficult, if not impossible.

1.5 Problem Statement

Complex Real World Simulations require huge processing powers, and costs. Simulations being built are scenario specific and there is minimum reuse of the existing capabilities. Therefore there is a need of a system which can enable us to carry out simulations with the existing use of resources, minimize Simulation development costs and maximize reusability. This definition of reusability leads to requirements of complying with a standard format. High Level Architecture is an industry standard currently being used all over the world for development of such simulations. So this infrastructure must be fully compliant with international standard High Level Architecture (HLA) and should be totally reusable and capable of integrating any federation/federate if that is HLA compliant.

1.6 Project Area

1.6.1 Distributed Simulation Platform

There are several reasons to choose a distributed computing architecture for simulating a ground combat scenario. By definition, distributed applications are modular and this design allows the simulations to be expanded and updated at any stage in the development. Modularity allows for a large degree of flexibility in the simulations since different programmers working on a wide range of platforms can create each of the different modules. Conforming to a standard for distributed computing will also allow future technological capabilities to be accommodated by the project.

A distributed environment allows the simulation to grow past the computational demands of a single machine. The distributed architecture acts as a common platform that joins different modules together and takes care of the interactions between them. Limiting the architecture to a single computer creates a computational ceiling the simulations would not be able to cross. A distributed architecture allows simulations to pass this limit as computation can now be spread to as many different machines as needed. In addition, different modules can require supporting software to be installed alongside the module. Distributing the modules across several machines gives the advantage of also distributing supporting software to multiple machines.

The most promising basis for such a standard is the HLA, a standardized middleware interface specifically designed for distributed simulations. The HLA is well specified and fully generalized. This has been concluded as per a detailed comparison of HLA with all Standards, Protocols and Techniques that can be used for such simulation scenarios.

1.6.2 Graphics

These days the emphasis is on high quality computer graphics in the field of Modeling and Simulation, M & S. Back end mathematical models are being supported by front end computer generated models, hence creating a very unique and realistic modeling environment. Be it the training of armed forces or missile testing, graphics plays its role in enhancing a user’s experience of the entire simulation. The two big standards being followed in the field of computer graphics these days are DirectX and OpenGL. Our framework provides us with the capability to achieve interoperability between disparate components, the graphical output of whom is covered by OpenGL. OpenGL is what we have employed in our project. Text based simulations are less user-friendly and are limited as far as their interactivity is concerned but with the application of OpenGL, our simulations provide the users a whole new world of interactivity. Hence our project scope involves the usage of OpenGL, Open Graphics' Library. [2]

1.7 Motivation

First and most important motivation that, simulations have become one of the most widely used techniques in operations research and management science. Despite its advantages, numerous problems are faced related to the computational ceilings imposed by existing hardware platforms. There is need to overcome this problem so that future developers are not constrained by the processor power to achieve the benefits of simulation. Secondly, it is the focus of most Developed Nations, and extensive research is under way, this gives a chance to participate in the ongoing international research.

1.8 Importance

The foremost achievement of the project is that it will provide Pakistan a giant leap towards self reliance in defence modelling and simulation. It has provided the first ever defence simulation that has HLA compliant RTI to Pakistan.

1.9 Sponsoring Organization/ Project Objectives

CARE- **C**entre for **A**dvanced **R**esearch in **E**ngineering is the sponsoring organisation for this project. Our project objectives are:-

1. To create a real time ground combat scenario and display it with state of art graphical techniques.
2. To allow user (both defender and attacker) to specify resources, options, domain and time limits, which will help application build objects.
3. On the basis of chosen options, create federates and federations.
4. Execute federations and achieve results.
5. Hold the statistics of different results and use it for future comparisons.
6. This process to be carried out in a real distributed environment using RTI.

1.10 Deliverables

1.10.1 Combat Simulation of 'Defence' Operation

Simulation of a typical gr cbt scenario consisting of various mobile fighting objects (tks, wpn heads, soldiers etc). Models of such objects, different parameters and conditions corresponding to real battle field ,and their simulation to view different results.

1.10.2 Interfaces :

These would be used to efficiently communicate between simulations and the RTI in a standardized format. They shall also provide language independence (currently for Java and C++ only). RTI Interface is the Interface that would be used by Simulations (Federates) to access the functionality of the RTI. Federate Interface is the Interface that would be used by the RTI to call back the functionality of the Federate.

1.11 Summary

This chapter has emphasis on the introduction of our area of interest, importance and motivation for the project, aim and background for selecting this project and then the problem statement based on which project objectives are finalized.

C h a p t e r   2

2. Design of the Battle

2.1 Introduction

Basically a particular scenario similar to plains of Punjab is selected. The design of the battle is planned as per the appreciation of the situation and courses of action available.

2.2 Terrain Information

The area represents terrain similar to that found in central and southern Punjab. It is generally flat and open providing good observation. The area is fairly well cultivated and there are clumps and rakhs which provide cover and at time limit observation. Villages are located on higher ground and provide good observation of the area. There are roads and tracks exist in the area. These are on raised embankments of 3 to 4 feet high. There are a number of kidney bunds in the area varying in height from 5 to 10 feet.

Obstacles include Ghazi DCB (Ditch cum bund) which is an anti tank ditch with 10 feet bund on home side. The ditch is 12 to 15 feet wide and 6 to 8 feet deep. It is charged with water from nearby canal and 3 to 4 feet water level can be maintained in DCB without significantly affecting the water level in canal. Other water channels are partial obstacle for tks. [3]

Meteorological conditions are the one which play very important role in any operation of war. Here in this simulation the actual met conditions are taken under consideration and it is being given below as table 2.1 [4]

Table 2.1 Meteorological Conditions

Event	**Timings**
First light	0500 hours
Last light	1900 hours
Moon	First quarter
Weather	Clear & Dry

2.3 Situation

Blueland and Foxland are neighbouring states with their interstate boundary along nullah. Recently the situation has worsened and both sides have ordered deployment of forces all along the border. Foxland bde with an armr sqn is reported to be assembling in north of Samba for a possible attk in Blueland area. Air is favourable to Foxland.

Blueland's Baloch regiment ex Blueland Brigade has moved to its forward location to defend its area of responsibility. CO Blueland's Baloch regiment has finalized his plan and called his 'O' group to give necessary instructions. Company commander 'A' company ex Blueland's Baloch regiment, was briefed that "We have indications that Foxland will commence hostilities with a sizeable effort anytime after D-day. Our peacetime preparations will pay dividends as per main defences on the Ghazi DCB are well prepared. However, to delay and attrite the enemy the Azizpur salient provides the most suitable terrain to do so. For this task (delay & attrite) available resources are, tp of tks, Field Engineers and arty Guns. Some of the en actions and own counter actions are described below in table 2.2. [4]

Table 2.2 Development of Operations (Enemy Actions & Own Counter Actions)

Enemy Operations /Actions	Own Counter Actions
(1) Enemy commences operation with a bde strength supported by sqn of tks by first lt D-day to reach main def by last lt D-day. Enemy is expected to use cbt avn and air power for the early clearance of security zone.	**(1)** Employment of covering tps so as not to allow the en to have a free run. Covering tps battle posns would be based on mutually sp def localities(kidney bunds) sp by minefields and will force the enemy to pre selected killing zone.
(2) En will use the air and gun ships to destroy our mobile elements. Enemy gr mnvr armr will attempt to avoid running battle with our covering tps and would aim at fixing and by passing it.	**(2)** Any attempt to bypass battle posns will be thwarted by fighting a successive battle, adjusting posns or moving to altn posns. Also request for close sp mns and cbt avn against en tks.
(3) On DCB, the en armr sqn will search for gaps and will attempt opportunity crossing. Failing to exploit the gaps the armr will get deployed on broad front and will engage the main defences.	**(3)** After the covering troops handover battle to the main defences they will occupy position along the DCB and in depth to cover the gaps and supplement the main defences.
(4) En will move from FAA on night D-day/D+1 and would reach in vicinity of main def before first lt D+1 and utilize D+1 for recce, probing attks.	**(4)** We will use/call upon additional artillery support , combat aviation and fighting patrols to deny enemy close observation of main defences.
(5) On night D+1/D+2 en will launch attk across the main def on DCB to make a bridge head. En will look for crossings on canal. En is likely to attk ‘A’ coy area of responsibility with a bn sp by a tp of tks.	**(5)** On determining the direction of enemy attack we will readjust defensive positions, call for artillery support and launch spoiling attack by fire or raid on likely enemy (FUP) forming up place.
(6) If enemy succeeds in breaching the main defences will try to consolidate his positions thereby reorganising along the ditch thereafter efforts would be made to hastily contact the depth positions.	**(6)** In case of loss of any position, launch a local counter attack. We will continue to provide on spot resistance on the DCB and cause enemy unacceptable attrition through all available means.

2.4 Plan/Course Adopted

To take up def position with two coys up along DCB. ‘A’ coy as lt fwd coy, ‘B’ coy as rt fwd coy, ‘C’ coy as lt depth coy and ‘D’ coy as rt depth coy. Covering tps comprising tp of tks, Listening Post and a Standing patrol ahead of DCB to provide early information and cause attrition and delay to en armr till he contacts main def. Bn HQ/res in area Azizpur. Listening posts ahead of forward coy's loc at far edge of protective minefield. Standing patrol in likely en FUP (forming up place) for spoiling attk. Def fire DF(SOS) for artillery and mortars along lt approach. Def fire (DFs) for artillery and mortars ahead of DCB all along the three approaches. Bias of anti tank defence towards left/western approach. Protective minefield ahead of main defences along DCB. Defence to be ready by first light(0500 hrs) D-day. Standing patrol to be in position by mid night D-day. Mine laying to be completed by first light D-day. Sketch for the Def deployment/Design is given below as figure 2.1. [5]

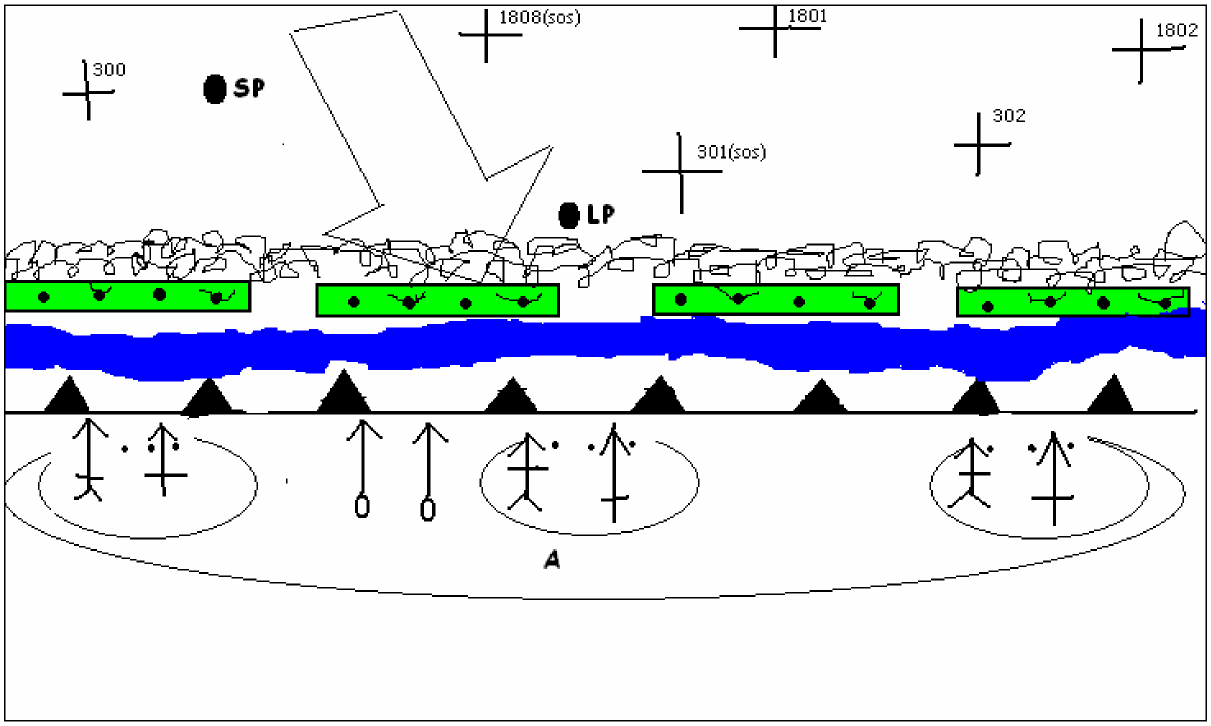


Figure 2.1 Defence Deployment / Design

2.6 Summary

This chapter gives us the understanding about the battle design, terrain info, situation of battle, en actions and own ctr actions, plan/course adopted of the battle.

Chapter 3

3. Terrain Module

3.1 Introduction

A Terrain Engine is a general purpose Tool to create and generate any type of outdoor scenery. The scenery should be convincing enough for a naked human eye to give illusion of a real outdoor scene. For a military solution, terrain, and a realistic one is one of the most essential components. The success of a terrain engine lies in its modifiability and the closeness of its behaviour to nature. At present the gaming industry holds one of the best terrain and outdoor scenery terrain engines developed for the flight simulators and the other tactical simulation games. However most of them do not have the capability to fulfil the requirements of a military simulation, as they are usually of static nature, and have fixed scenarios. With this aim effort was made to make a general purpose terrain engine fulfilling the basic requirements of a military simulation. [6]

3.2 Parts of a Terrain Engine

A terrain engine generally consists of parts, like terrain generation, trees, Foliage, Water Bodies ,Clouds, bunds, Marshy area and ditches. [6]

3.2.1 Terrain Generation

The terrain generation has been carried out using a height map. A height map is a set of grey scale colour values that determine "height." Here height map from a .raw file was read then a texture was applied over the entire terrain. The terrain is rendered using triangle strips as shown in figure 3.1. To tile a second texture on top of the first one for giving the appearance in more detail. This is called detail texturing, which can add a great deal of realism to a scene. Multi texturing is used to achieve this neat effect.

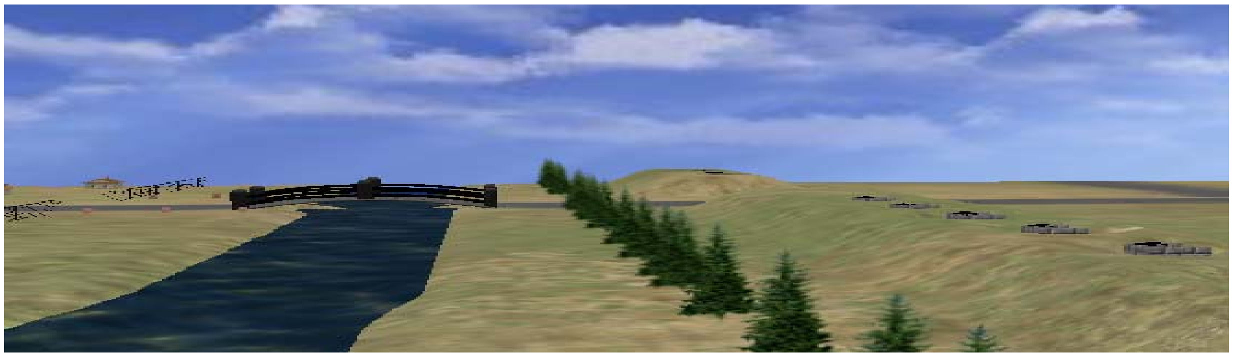


Figure 3.1 Snap Shot of Terrain

First job is to read the height map from the .raw file. This is simple because there is no header to a .raw file; it is just the image bits. This file format isn't what individuals generally want to use because to either know what the size and type are, or guess. GL_TRIANGLE_STRIP was used for the purpose. This means that there is no need to pass in the same vertex more than once. Each 2 vertices are connected to the next 2. To do this in one strip, there was a need to reverse the order every other column. It's like moving the lawn. Go to the end and turn around and come back. If it is not done this way, individual will get polygons stretching across the whole terrain. Multitexturing was added so that a detailed texture could be applied over terrain. This gives the terrain a more detailed look, instead of a stretched look. To do this, normal Multi texturing functions was used, but detail texture's properties were changed with GL_COMBINE_ARB and GL_RGB_SCALE_ARB. These 2 flags allowed increasing the gamma on the detail texture so that it doesn't over power the texture of the terrain. The last thing was to fiddle with the texture matrix. Instead of calculating the (u, v) coordinates for the tiled detail texture, it was just assigned the same (u, v) coordinates as the terrain texture (the whole texture stretched over the terrain), then scaled the texture coordinates by entering the texture matrix mode and applying scale value. It eventually wraps around again. Further improvements include, calculating colour of each vertex as per its height. So Giving 4 bands of colours like underwater, low range, mid range and high range, simply colours the whole terrain in beautiful shades as per height.

3.2.2 Blending / Masking

The root of all Terrain features discussed is a simple technique called blending and masking. In both part of an image drawn is not shown and becomes transparent. Alpha values are specified with glColor*(), when using glClearColor () to specify a clearing color and when specifying certain lighting parameters such as a material property or light-source intensity. The pixels on a monitor screen emit red, green, and blue light, which is controlled by the red, green, and blue color values. When blending is enabled, the alpha value is often used to combine the color value of the fragment being processed with that of the pixel already stored in the frame buffer. Blending occurs after your scene has been rasterized and converted to fragments, but just before the final pixels are drawn in the frame buffer. Without blending, each new fragment overwrites any existing color values in the frame buffer, as though the fragment were opaque. With blending, one can control how (and how much of) the existing color value should be combined with the new fragment's value. Color blending lies at the heart of techniques such as transparency, digital compositing, and painting. The most natural way to think of blending operations is to think of the RGB components of a fragment as representing its color and the alpha component as representing opacity. Transparent or translucent surfaces have lower opacity than opaque ones and, therefore, lower alpha values. For example, if one is viewing an object through green glass, the color seen is partly green from the glass and partly the color of the object. The percentage varies depending on the transmission properties of the glass: If the glass transmits 70 percent of the light that strikes it (that is, has an opacity of 20 percent), the color seen is a combination of 20 percent glass color and 70 percent of the color of the object behind it. One can easily imagine situations with multiple translucent surfaces. [7]

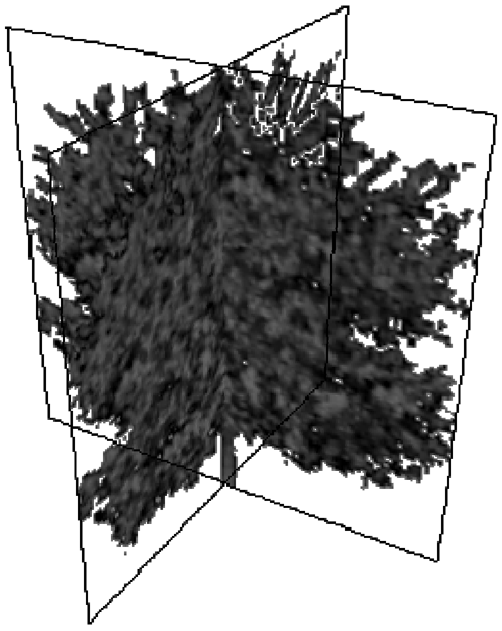


Figure 3.2 Cross-Section View for 2D Tree Image

3.2.3 Trees /Foliage

At present the industry level terrain engines with the help of strong graphic cards are producing trees as 3d structures, but a simple approach was adopted with out sacrificing too much visual accuracy. As shown in figure 3.2. The Tree is nothing but two quads over which alpha blended or masked image has been pasted. The result is however neat. With this low polygon solutions trees were drawn at a reasonable frame rate. These trees are covered in the entire area . Where ever go, one can find trees scattered there randomly, as they move along with the user. Trees are placed to create an accurate environment required to create a natural looking terrain.

3.2.4 Water Bodies

These are created with tillable textures. A total of Only 9 textures being used for animation, giving a smooth shimmering water effect.

3.2.5 Clouds

These move in 2 layers circling the area, moving at a very slow pace. Clouds are very important feature in real time situations. For optimization 2D cloud generation carried out. Distance and angle of presentation give illusion of 3d clouds. [7]

3.2.6 Bunds / Marshy area

We also have depicted some kidney bunds and two marshy areas on right approach. The main bund in our terrain is the DCB. Where as rest of the Bunds have height varying from 5 to 10 feet. The marshy areas are encircled by the clump of tree. One marsh is in between DCB and nullah. Where as other marsh is in the enemy area near international border.

3.3 Summary

In this chapter we have depicted that how the terrain module is being generated. Basically, first of all .RAW files were prepared in adobe photo shop then these files were loaded in the OpenGL code. Next step was the multi-texturing to give the effects of reality. Last step was to load the necessary models/ textures which are necessary for the terrain to present a natural look.

Chapter 4

4. Camera Module

4.1 Introduction

For the orientation of terrain there was a need to use a "camera", so that one can view 3D scene in all the directions. First of all setup the view volume and initialize the camera. In OpenGL there is only one library (glu) that provides a camera functionality i.e. gluLookUp (). But it can’t directly be manipulated to get the 6 degree of freedom. Therefore, that class of camera was used which could met all necessary requirements.

An OpenGL camera consists of three vectors: position, view and up. The **"position"** is the actual point where the camera is located, while the **"view"** is the target point that the camera is looking at. The position point and the target point form a view-vector. The **"up"** is the point that decides if the camera is tilting. All three vectors are shown in figure 4.1.Where as the view volume of camera can be set up by establishing the view angle in y-axis and in x-axis to show the far clipping plane and the near clipping plane.

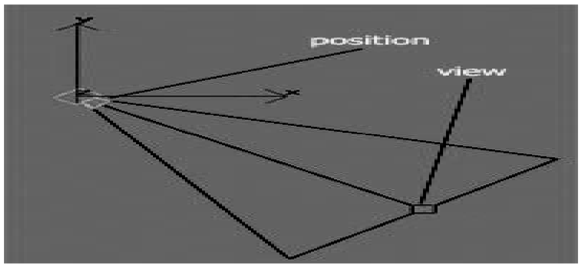


Figure-4.1 Camera View Volumes

4.2Camera Control

OpenGL camera control is somewhat unintuitive at first glance. The idea behind OpenGL is that it manipulates every input vertex using two user-specified 16-element matrix transformations. So as to achieve 3D perspective, or to achieve rotation. Combinations of these are used to produce camera control.

The two matrices in OpenGL are the **ModelView** and **Projection** Matrices. In order to change the settings, one has to tell OpenGL that one wants all future function calls to affect the PROJECTION matrix or MODELVIEW. Various OpenGL functions then can be called that set up or modify the camera -- **glPerspective, glRoatate, or glTranslate**. [8]

4.2.1 Function gluLookAt()

The gluLookAt function call provides many functionalities. Some of them are listed within this paragraph. In application program three things are stored: eye position, the point in space that the eye is pointed at, and the orientation of the eye (the up vector). OpenGl internal matrices can be used and the user specified matrices to move these numbers around to achieve whatever effect is wanted for instance, rotation or translation. The idea is that, to pass these 9 variables (3 per coordinate, three groups of variables) to the gluLookAt function, which sets up the current transformation matrix so to get the proper view in the scene.

This approach requires matrices to be handled at user level initially then these matrices are left to OpenGl for further calculations. Every change to an OpenGL matrix is actually cumulative. To change a matrix in OpenGL, the current matrix is multiplied by some new value, to produce a new matrix. This means that in this camera changes are cumulative. OpenGL provides ways to limit the effect of a multiplication, or rather, undo its effect, using a stack: the current matrix can be pushed onto a stack before it is altered. At any time, the stack can be popped up to return the stored matrix state. This is quite useful for camera control. A summary of OpenGL camera control commands is given in table 4.1.

Table 4.1 Camera Control Commands

Command	Description
glLoadIdentity	Initializes the scene.
glMatrixMode	Move control to camera or model
gluLookAt	Used to position camera in scene
glRotatef	Rotates everything after the call by an amount about specified axis
glTranslatef	Moves things in x, y, z
glLoadMatrixf	Loads matrix in OpenGL as transformation matrix
glPushMatrix	Saves current camera values
glPopMatrix	Restores the previous camera values.
glGetFloatv	Used to Query the OpenGL about the current matrix values
gluPerspective	Sets the view volume of camera

4.2.2 Description of Camera Control Commands

The **glLoadIdentity()** Sets the currently modifiable matrix on the top of matrix stack, which have been selected by user to work upon, to the 4 × 4 identity matrix.

The **glMatrixMode(mode)** Specifies whether the later operation will affect the modelview matrix or the projection matrix, using the argument GL_MODELVIEW, GL_PROJECTION for mode(within parentheses). Subsequent transformation commands affect any one of the specified matrix. Note that only one matrix can be modified at a time. By default, the model view matrix is the one that's modifiable, and all the matrices contain the identity matrix.

The command **gluLookAt()** defines a viewing matrix and multiplies it with the current matrix as under.

gluLookAt **(** GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz **)**

The desired viewpoint is specified by eyex, eyey, and eyez. Where as the centerx, centery, and centerz arguments specify any point along the desired line of sight. The last three arguments upx, upy, and upz indicate which direction is up ie tilt.

The **glRotatef** and **glTranslatef** functions are used to rotate the scene by a specified amount in terms of angle in degree about the specified axis and translate the scene by a specified amount along the specified axis respectively.

The **glPushMatrix** function pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on the top of the stack is identical to the one below it.

The **glPopMatrix** function pops the current matrix stack, replacing the current matrix with the one below it on the stack.

The **glLoadMatrix{fd}(*const TYPE *m)*** Sets the sixteen values of the current matrix to those specified by m. The values may be user specified values later on calculated by OpenGl. [9]

The **glGetFloatv**(GLenum pname, GLfloat * params) returns values for simple state variable. It is used to query OpenGl about value of current matrix. The pname parameter is a symbolic constant for the state variable to be returned, and params is a pointer to an array of the indicated type(4x4 matrix) in which to place the returned data.

The **gluPerspective** (GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far) Creates a matrix for establishing the perspective-view volume and multiplies the current matrix by it. Fov is the angle of the field of view in the x-z plane as shown in figure 4.2, its value must be in the range [0.0, 180.0]. Aspect is ratio of the view volume, its width divided by its height. Near and far values the distances between the viewpoint and the clipping planes, along the negative z-axis.

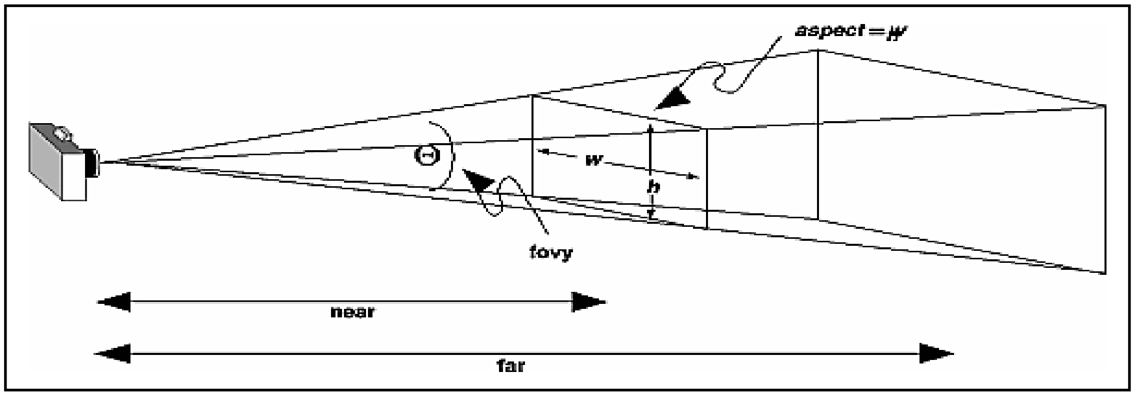


Figure 4.2 Camera Field of View

4.3 Rendering Transformations

Some transformations were required to show models on screen. Basically there are three sets of geometric transformations i.e. **M**odelling transforms, **V**iewing transforms, **P**rojection transforms. Basic model coordinates are shown as in Figure-4.3.

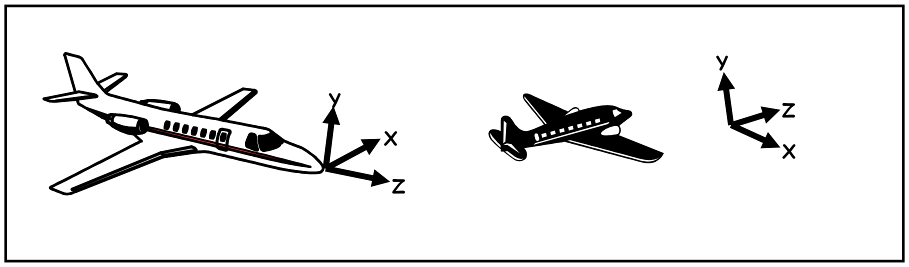


Figure 4.3 Basic Plane Coordinates

4.3.1 Modelling Transforms

In modelling transforms the size of the model is dealt with, its place is selected in the scene; model is scaled (to enlarge or reduce its size), rotated and translated.

4.3.2 Viewing Transform

In viewing transform models are not dealt with alone. But the complete scene which contains the models is Rotated & translated so that the world lie directly in front of the camera. The viewing transformations is carried out normally in two steps, which are typically place camera at origin and typically looking down -Z axis

4.3.3 Projection Transform

In Projection transform for 3D scene we use the perspective projections transforms and for 2D scenes the ortho transformation is carried out. Camera eye projection view is shown in Figure 4.4.

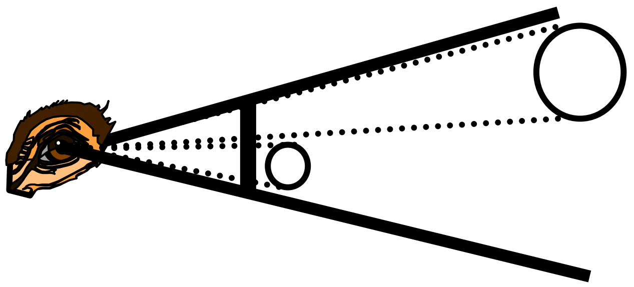


Figure 4.4 Eyes Projections View

4.4 Camera Terminologies

Terminologies related to camera are **grand/global** (See Figure 4.5 and 4.6) or a reference coordinate system which remain constant throughout the transformation and a **local** coordinate system (see Figure 4.7 and 4.8) which changes with each transformation.

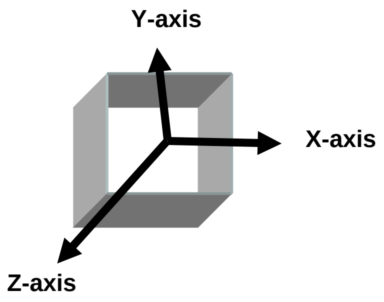


Figure 4.5 Basic Global Axis

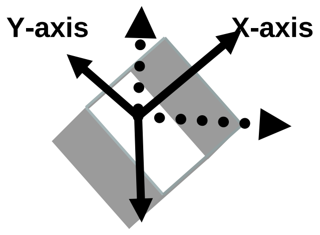


Figure 4.6 Global Axis Rotation

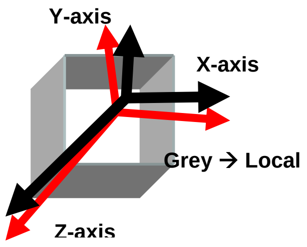


Figure 4.7 Local Axis

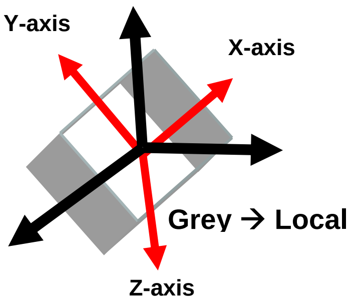


Figure 4.8 Local Axis Rotations

Pitch is Rotation about local & Global x-axis. **Yaw** is Rotation about local & Global y-axis. **Roll** is Rotation about local & Global z-axis. **Move Forward/Backward** is any movement along + Z or – Z-axis. **Raise/Lower** is any movement along + Y or – Y-axis. **Strafe Right/Left** is any movement along + X or – X-axis. **6DOF**--Degree of Freedom is Any Camera which is capable of performing all the above mentioned transformations is said to have achieved 6DOF.

4.5 Setting Up a Camera in OpenGL

For the effective use of camera, the steps are as under:

4.5.1 Creating a transformation matrix.

4.5.2 Selecting Matrix Mode i.e. GL_PROJECTION using glMatrixMode().

4.5.3 Setting the view volume using gluPerspective().

4.5.4 Selecting Matrix Mode i.e. GL_MODELVIEW using glMatrixMode().

4.5.5 Saving state of matrix on top of modelview matrix, stack using glPushMatrix().

4.5.6 Loading transformation matrix as modelview matrix using glLoadMatrixf().

4.5.7 Carrying out required transformation using glRotatef() glTranslatef().

4.5.8 Queering the OpenGl about the current modelview matrix values and loading them into transformation matrix using glGetFloatv().

4.5.9 Restoring the state of saved modelview matrix using glPopMatrix(), Updating the Scene, Rendering the scene.

4.6 Simple Movement System

Matrices are used to represent the coordinate system for camera movement. To build a simple 3D 6DOF movement system, the essential features are moving and rotating. Thus the basic camera parameters are shown in Figure 4.9.

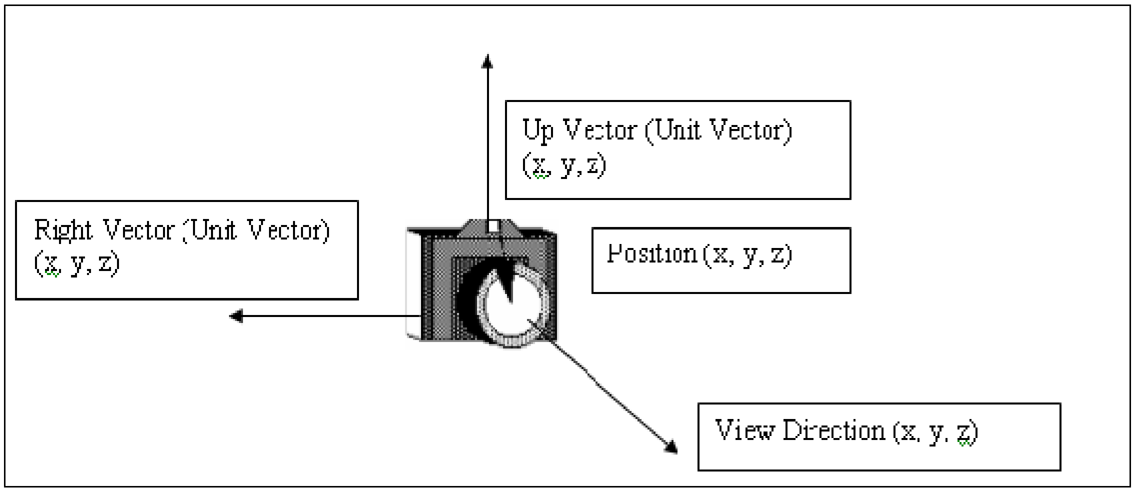


Figure 4.9 Camera’s Parameters

Camera orientation is shown by three vectors: The *view direction*, the *right vector* and the *up vector*. Initial position of camera is shown by the position vector. *Viewpoint* is achieved by the addition of position vector and the view direction vector. Initially the orientation points along the negative z-axis: View Direction (0|0|-1), Right Vector (1|0|0), UpVector (0|1|0) and the position vector (0|0|0).These all vectors have been incorporated in the transformation matrix. The last line of the transformation show the position of the camera that always lie on the origin of the current coordinate system. Transformation matrix formed from these initial vectors is shown as table 4.2.

Table 4.2 Transformation Matrix

1 0 0 0
0 1 0 0
0 0 -1 0
0 0 0 1

4.6.1 Movement Along All Axes

Let's say, if one uses the function call, MoveZ(float distance), then this movement is related to translation along local z-axis only by an amount distance. However, the movements related to translation like along x-axis, y-axis and z-axis have been dealt with using two different approaches which are described later within this paragraph. The function calls necessary for movement along all axis are given in table 4.3.

Table 4.3 Function Calls for Move along all Axis

glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadMatrixf(Transform);
glTranslatef(0,0,distance);
glGetFloatv(GL_MODELVIEW_MATRIX, Transform);
glPopMatrix();

In **first approach** OpenGL functions have been used to carry out the translations about the specified axis. First of all , OpenGL has been told about the current matrix ie make model-view matrix of current matrix, so that all subsequent transformations affect the model view matrix then current state of model-view matrix have been saved so that the effect of subsequent operations is not cumulative. Then transformation matrix is loaded on current stack of model view matrix. After loading the transformation matrix , translate operation is carried out. This translation effects particularly the transformation matrix because it is multiplied with the current matrix (The current Model View matrix).Now after translation have occurred , the values of transformation matrix have been changed , representing another coordinate system. OpenGL returns the current values of the model-view matrix which are then stored in transformation matrix. Previously stored model-view matrix is restored then. After all this, camera is updated in each frame using update() function. After the camera is updated translated scene is rendered using the Render() function, being shown in table 4.4.

Table 4.4 Render Scene Function

void CCamera::Render() {
ViewPoint = Position + ViewDir;
GluLookAt (Position.x,Position.y,Position.z,
ViewPoint.x,ViewPoint.y,ViewPoint.z,
UpVector.x,UpVector.y,UpVector.z);}
Transform[12] += Transform[8] * distance;
Transform[13] += Transform[9] * distance;
Transform[14] += Transform[10] * distance;

In **second approach** by the use of translation operation, since only the last row of transformation matrix is affected, hence only the last row is simply added into the specified direction vector and then multiplied by the amount of distance to be travelled along particular axis e.g. if the move is required along view vector that is represented in programme by the row Transformation [8], Transformation [9], Transformation [10].

Now if the function MoveZGlobal(float distance) is called, this camera will move along Global z-axis instead of Local z-axis as in previous case. This function is using the formula that, if move is required along global axis, the movement will be independent to that of local axis. The value of transformation matrix just incremented along the movement axis E.g. to move along global z-axis Transform [14] will be incremented, and movement along global z-axis is achieved. Using this approach a movement can be made along specified axis by any amount of distance.

4.6.2 Rotation About All Axis

If we call the function RotateZLocal (float deg), the view is rotated about Local z-axis. In other words the view has been rolled. This function also uses logically arranged OpenGL functions to carry out the Rotation about the specified Local axis. First of all, OpenGL has been instructed about the current matrix i.e. make model view matrix as current matrix, so that all subsequent transformations affect the model view matrix. Then the current state of model view matrix has been saved so that the effect of subsequent operations is not cumulative. Then transformation matrix is loaded on current stack of model view matrix. After loading transformation matrix, Rotation operation is carried out. This Rotation affects particularly transformation matrix because it is multiplied with the current matrix (The current Model View matrix).Now after translation have occurred, the values of transformation matrix have been changed, representing another coordinate system.

OpenGL returns the current values of the model-view matrix which are then saved in transformation matrix and transformation matrix is updated. And previously stored model-view matrix is restored. After all this, camera is updated in each frame using update() and then the scene is rendered using Render() function.

4.6.3 Mouse Integration

The function SetViewByMouse() is called whenever the mouse is moved. This function is integrating the mouse with camera and gives the control to the mouse whenever the mouse is moved. In this function a window structure POINT that holds the X and Y coordinate is used. Then the binary shift operator has been used to get the mid point of window height and width. Then the mouse's current X, Y position using GetCursorPos() is obtained .Then camera's local and global functions are obtained to rotate the view with the help of mouse.
Zooming effect can be achieved by manipulating the first argument of gluPerspective() function of OpenGL, which is used to establish the view volume in OpenGL.

4.7 Camera Features

This camera has many features other than listed in this document. The main features which have been used/implemented in project are Roll, Pitch, Yaw, Move Forward/Backward, Move Up/Down, Strafe Right/Left, The global mode features are Roll, Pitch, Yaw, Move Forward/Backward, Move Up/Down, Strafe Right/Left.

4.8 Summary

This chapter describes the camera functionality in detail along with the salient features of the camera. Certain common terminologies used in manipulation of camera are also touched upon to give the basic knowledge of working of the camera. We have discussed in detail each and every function being used by OpenGL to manipulate the camera.

Chapter 5

5. Tank Module

5.1 Introduction

Basically Tanks play very important role in the battle field. Pak Army has number of Armour Regiments With variety of tanks available. Tanks are normally part of fighting elements but these can play very effective role in defence as well.

Here in gr cbt scenario we have tanks on both sides i.e. foxland and blueland. In def these tanks will be taking part in the delay battle against enemy and then will ultimately fall back to main defs. These may also be used for ctr attks to get back the lost piece of gr from enemy. Where as enemy will be using these tanks for assault on main defs, as shown in figure 5.1. This figure shows the view of en tanks formation. Different color schemes were used to show the difference between enemy(foxland) and own(blueland) Tanks.

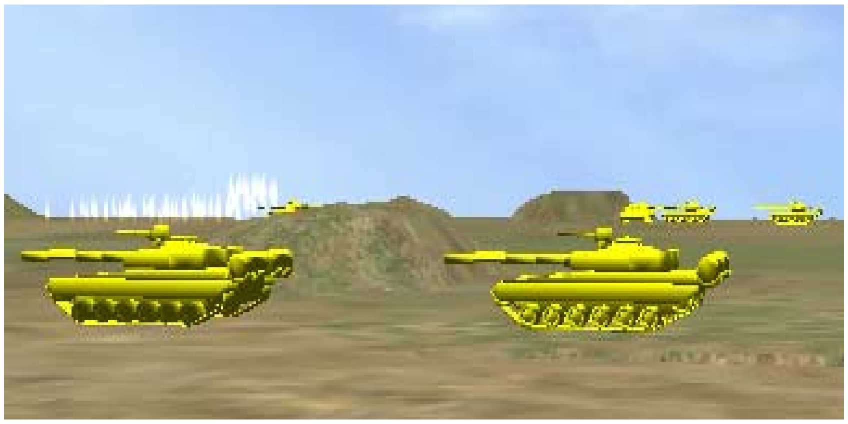


Figure 5.1 Enemy Tanks View

Tank module is being constructed with the help of polygons. It is divided into different parts like turret, barrel, gun and body etc.

5.2 Tank Modelling

Tank being used in this simulation is **.3ds** model. We have number of polygons in one model of tank. The greater number of polygons affects the efficiency in terms of performance. If we apply the tank and its instances as one complete unit in the simulation then naturally it would have adverse affects on performance and speed. So to cater this problem a software **Accu Trans** was used**,** which divides the complete model into different parts. [6]

5.3 Accu Trans 3D

It reduces the number of polygons and enhances the overall efficiency of the simulation in terms of performance and speed etc. One gets another advantage of working on fine details of tank like its gun movement, turret movement etc. This model is being divided into parts like; Barrel, Turret, Machine Gun and its Body (incl chain, engine etc)

Accu Trans also provides accurate translation of 3D geometry between the file formats used by many popular modeling programs. Materials attributes are transferred between the files. Textures are supported.

5.4 Tank Mechanics

The process of tank movement is very difficult and time taking task. However, the complete movement and rotation of the tank and its instances was successfully managed. Rotation of the tank is now 360 degrees. Similarly, the rotation of the tank turret and its barrel are also successfully done. Turret and barrel rotate as one unit around 360 degrees. Machine gun can also move with turret movement. But the moves of turret, barrel and machine gun are not implemented in this simulation due to certain short comings.

As far as tank movement is concerned it is also done so that tank can move freely forward, backward, turn right/left while at static position as well as during move.

5.5 Tank Response to the Terrain Features

As already discussed, that the terrain and tanks are separate module. Both of these are integrated together along with many other small models to form a comprehensive simulation. However, there were certain problems during integration. The issue of tank response to the terrain features had to be addressed like its response to the bumps, nullahs, ridges, fence, buildings, forests and marshy areas etc.

Tank response to the bumpy areas is clearly visible when it moves through ground. When tank moves through water then sound to that effect is also produced. Previously we had a problem that tank during move used to pass through ridges etc. But now when it senses any such obstacle ahead then it moves over that ridge. Tank response to the marshy areas is also worked out, where it bogs down into the marsh.

5.6 Tank Firing

Another important aspect is to show the tank firing which is also being accomplished. Although all aspects of firing (like aiming, calc range, loading rounds etc) are not covered due to the shortage of time. Only the firing of rounds towards the target is shown.

5.7 Summary

The tank module is being produced with lot of efforts along with its mechanics. We have designed to types of tank controls i.e. Manual and Automated. The manual control is in the hands of user and he/she can move the tank any where and in any direction within the given terrain area. Moreover, we also have depicted the tank firing which is not worked too much of the details. Tank response to the terrain features is also part of this module development; however, complete collision detection is not implemented due to the shortage of the time.

Chapter 6

6. Use of Bomber Aircraft and Missile

6.1 Introduction

Weapons technology has most apparent advancements in the field of air threats. Specially Gr cbt elements are vulnerable to the bomber air attks. In the next few years Gr cbt forces must have advances against air threats of air-to ground missiles, attack helicopters, drones, tactical aircraft and unmanned bombers. High performance avionics and multifarious attack profiles make the Unmanned Bomber most furious.

We have shown that enemy enjoys air superiority and likely to use bomber aircrafts. So the bomber aircraft model is loaded as part of simulation along with the attk of missiles.

6.2 Calc Flight Path of Aircraft

Calculation of flight path of Bomber aircraft has to be done while keeping certain factors in mind. There are many factors that effect the flight path but the factors related to gr cbt or gr tgts(Target) may include effect Of Tgt on Flight Path and Physical Shape of Target. Bomber aircraft must achieve the required level of destruction of the target. Targets like bunkers on DCB, important roads, bridges, villages, military installations and water channels etc are such that they require special techniques and direction of attk to acquire the required level of destruction. For example, Bunkers on DCB, require air craft to come from a direction which is along the axis of the Bunkers on DCB. If air craft attks the bunkers on DCB from a direction perpendicular to the direction of the bunkers on DCB , then all the planning and the effort made is useless. However this calculation can only be done by the pilot of manned bomber or the sender of unmanned Bomber aircraft, and till now no technique has been developed to calculate the effect at run time for the unmanned bombers.

6.3 Type of Target

Target Type is also one of the important factors for adopting the specific attk profile/technique. Ammunition dumps, Nuclear Installations etc are some types of the tgts which require special attk techniques. For such tgts the aircraft has to come at a high altitude to maintain its own safety. If it makes a lower attack profile to attk such tgts, ultimately it will also be damaged due to massive destruction of the tgts.

6.3.1 Targets Location

Within types of tgts, Moving or Stationary targets definitely effect the type of weapon/missile to be used and the flight path to be followed .Moving targets like Tanks/vehicles require that the flight path to be in the direction of move of the tanks/vehicles. More over the formation of the moving tanks also effects the direction of attk. If the flight path is in the perpendicular direction to the direction of move of tanks/vehicles convoy then maximum destruction cannot be achieved. The release of weapons/missiles at particular time, angle and direction also matters a lot. If this is not done so then there is a chance that a fast moving tgt may get out of the dangerous zone of the weapon/missile thus resulting in reducing the hit probability and missing the objectives. Tgts like tanks/vehicles may require a low level attk techniques. However, High altitude bombing may be carried out on the tank concentration areas as the accuracy of the guns mounted on the tanks increases when the tanks are static. Stationary targets in the battle on field like Artillery concentration area and deployment areas have a specific shape. Bunkers DCB also have specific layout. So the aircraft is bound to attk from a specific direction to achieve maximum destruction of the tgt.

6.4 Type of Weapons/Missiles

Types of weapons/missiles have direct bearing on the flight path of the Bomber. Different weapons dictate different attack profile to be adopted for their delivery. Some require high altitude attack profiles while others require low and medium altitude attack profiles for their delivery .The attack profiles are necessary for the weapons to achieve maximum destruction out of weapons. The effect of particular weapon/missile is as follows.

6.4.1 Bombs

For the delivery of cluster bombs, initially the air craft has to fly at an altitude of approximately 200 meters .When the aircraft is at a distance of 600 meters, it releases its weapons.

Where as for the delivery of Napalm bombs, initially the air craft has to fly at a very altitude of 50 meters .In case of Napalm Bombs, the air craft releases its weapons when it is at an approximate distance of 250 meters from the target area. After it has released its weapons on the target, it flies back maintaining the same height.

When the weapons are Ballistic Bombs, the aircraft comes at an altitude of 1500 meters to attack the tgt. When the aircraft is at an approximate distance of 3000 meters from the tgt, it aligns its direction with the tgt making an angle of 30 degrees with the horizontal and then starts diving down towards the tgt. When the aircraft is at approximate distance of 1700 meters from the tgt, it delivers the weapons for achievement of its objective.

6.4.2 Guns

We do have guns mounted within the bomber/fighter aircrafts. So if it has to use guns for attacking the gr forces then the aircraft again comes at low altitude to attk the tgt. When the aircraft reaches at approximate distance of 2500 meters from the tgt, it and aligns its direction with the tgt making an angle of 10 degrees horizontally and then starts diving down towards the tgt. When it reaches at an approximate distance of 1700 meters or at a time distance of approximate 3 seconds, it delivers the weapons for the achievement of desired destruction level.

6.4.3 Rockets

If the weapons used are rockets, the aircraft must come at low altitude to attk the tgt. When the aircraft is at an approximate distance of 3000 meters from the tgt, it aligns its direction with the tgt making an angle of 20 degrees horizontally and then starts diving down towards the tgt very smoothly. During dive when reaches at an approximate distance of 2000 meters or at a time distance of approximate 2 seconds from tgt, which may vary with speed of Bomber, it delivers the rocket. But just after delivering it moves back to its take off place.

6.4.4 Missiles

When the weapons are Missiles, the aircraft comes at an approximate altitude of 700 meters to attack the target. When the air craft is at an approximate distance of 2500 meters from the tgt, aligns its direction with the tgt making an angle of 20 degrees horizontally and then starts diving down towards the tgt. When the aircraft is at an approximate distance of 1000 meters, it delivers the missiles for the destruction of tgts.

6.5 Effect Of Time On Air Attk

The time of air attk is very much important to decide the particular attk profile/technique. During the night hours the attacker has to totally change the attk strategy. Where as in day light the direction of sun plays an important role in dictating the flight path of Bomber or any air craft. An attack coming from the sun, or going into sun after attack, is bound to make things more difficult for the defenders. More over weather may be rainy and foggy. So in such cases without the use of radars, detection of aircraft may be very difficult.

6.6 Effect Of Terrain Features

Terrain features also have their own role in dictating the flight path of Bomber. For example, in Hilly area, the Bomber has to move with care to avoid any sort of hit with any terrain feature. Similarly, while passing by a thick jungle, a low flying Bomber has to adjust its flight path to avoid collision with tree tops. A Bomber flying in plane grounds may be visible from a large distance and may find enemy guns and missiles ready to engage it once it reaches at target for delivery of weapons.

6.7 Implemented factors

Many factors effect the flight path and attk technique of any bomber aircraft. However, out of all above complexity factors, the factors chosen to be incorporated in the project were Type of weapon (i.e. missiles), Type of target (i.e. tanks) and to cater for terrain features. Incorporation of other factors is not possible due shortage of time.

6.8 Implementing Aircraft Shape/Behaviour

The main problems encountered were accurate shape, behaviour, frame rate etc. The problems faced in simulating behaviour of a bomber are:

6.8.1 Accurate Shape

For the shape accuracy of the aircraft we have used 3d model with as much number of polygons so that both the accuracy and other details like performance and real touch are not affected to greater extent. Moreover, unless the aircraft's shape is closer to the real object being simulated, the behaviour cannot be simulated accurately, hence a true representation is must.

6.8.2 Accurate Behaviour

Behaviour of any aircraft has dynamic range of certain factors to be considered. Behaviour may include speed, direction, angle, altitude and weapons load etc. For example, when an aircraft moves with a specific speed, and its movement should also well defined in terms of angle of turning and elevation, rate of turning and climbing (altitude) etc.

6.8.3 Effects Of Computer Performance

Keeping in view the speed of processor, RAM, hard disk space etc, the same simulation may run with varying speeds on different machines. This problem unless addressed properly will pose a big problem which may hinder the creation of true representation for accurate flight paths. Each system can show the path being traversed at a different time rate.

6.9 Implementation Details

First of all a model made available in 3D space, able to be placed anywhere, thus giving real look as well as better performance. A work on basic movements of a stationary model, i.e. heading and move back etc are also done.

Universal frame work for a path to be traversed at a constant rate on all types of different machines, using system clock ticks, and fps based calculations. Movement are achieved in 2D plane, with fixed height. Framework for pre-defined path of fixed aircraft movement was decided.

Planned behaviour includes only start point and location of target (stationary) given as input. Rest all will be calculated by mathematical equations used for its movement. General missile firing profile characteristics are fed as guideline in calculating flight path. Aircraft will follow the path with accurate heading, elevation angles, attack angles and accurate speeds. Missiles will be fired by the aircrafts at the general area of DCB. Three aircraft models were used as an air attk force from enemy side.

6.10 Summary

This chapter is most important with a viev that it provides us the complete picture that how all the aircrafts were created and utilized in an efficient manner to produce a air attk simulation. Detailed description for each model is also shown along with figures in chapter 8. Moreover, the missiles firing is also worked out using Bezier curves equation.

Chapter 7

7. Sound Effects in the Simulation

7.1 Introduction

We have made use of FMOD to create the sound effects of different elements of war as well as nature. FMOD is cross platform audio engine, It can be easily used and can be made available on the Windows, Windows CE, Linux, Macintosh, GameCube, Playstation 2 and XBox platforms. It can be used with C/C++, Visual Basic, Delphi and MASM. So, if you use one of those languages on one of these platforms and you want to use sound in your application, FMOD is made for you.

7.2 Alternatives

One alternative might be OpenAL. OpenAL is another cross platform audio API which is available for Windows, Linux and Macintosh and can be used with C/C++, Delphi and Java. The style is similar to OpenGL (for example, the extensions technique is also used here). Where as a Windows-specific alternative would be DirectSound which is part of Microsoft's DirectX.

7.3 What Does it Cost?

The developers of FMOD have a nice philosophy; if you don't intend to make any money with your project, you can use it for free. So as long as you don't make any profit with your program you don't have to pay anything. To use FMOD in your application you need the headers and the library.

7.4 Choice among FMOD or OpenAL?

Here in the simulation, the FMOD audio engine was choosen as better choice with lot of variety. FMOD is subdivided in two APIs; FSOUND and FMUSIC. Here FSOUND API was used, which include the samples and streams. Mostly sample sound effects were used at the background of the simulation.

However, just for the purpose of the knowledge this chapter also has some information about other API (FMUSIC) of FMOD Engine. On the other hand, although only visual C++ was used, but the information about others users like Delphi and visual basic etc is also given.

7.5 Getting Started

After unpacking the archive of FMOD, we copied the FMOD.DLL into our working directory. It is also better to copy it into the directory where executable file are or will be located. Before we can really start we have to do following things as per the requirement of specific user. For C users one only has to include the header "fmod.h" and depending on compiler, the right import library among one of the following: Within the code “fmodvc.lib” was used for Microsoft Visual C++. Where as for Delphi Users only include the unit FMOD in the uses clause. For Visual Basic users there is need to add “fmod.bas” to the project.

7.6 Initialization

Before the FMOD to play some sounds for us there is need to initialize it as well. This is pretty simple and can be done by following line of code only:

```
FSOUND_Init (44100, 32, 0);
```

The first parameter is our output rate in hertz. In obove example it's equal to 44100. The second parameter is our maximum number of software channels. It doesn't matter if you choose a higher number due to the fact that it won't affect your CPU usage as long as you don't really use them. However 32 should be more than enough for this short introduction. In the third parameter can we specify some flags if we want. We leave this parameter as 0. That's it. Now it’s ready to play some sounds. But which format of sounds to be used? Is it a song, a sample or a stream?

7.7 Introduction - Songs, Samples and Streams

Now it's time to mention that FMOD is subdivided in two APIs; FSOUND and FMUSIC. Which one you should use depends on the files you want to play. However, I have already mentioned that on our simulation we are just using FSOUND API for playing the background samples. They can be treated as either samples or streams. If you want to play a short sound / a small file like a gunshot then you treat the file as a sample. Samples will be decompressed into memory before being played and can be played multiple times. When we want to play a bigger file like continuous background music then we handle the file as a stream. This will result in more CPU usage because the file will be streamed from the disk in real time, but on the other hand it also needs less memory as a result. Another thing to note is that streams cannot be played multiple times at once.

7.8 FSOUND

Due to the fact that we have more and better possibilities while using FSOUND it's a little bit more complicated. For instance, we have several channels which we can use simultaneously. But in this short introduction we will only be using one. Within FSOUND we have samples and streams. First we have a look at samples, and afterwards we move on to streams.

7.8.1 Samples

7.8.1.1 Sample’s Handle

To play a sample we needed a FSOUND_SAMPLE variable for the handle and following two lines did the job:

```
handle=FSOUND_Sample_Load (0,"YourFileName",0,0,0);
FSOUND_PlaySound (0,handle);
```

The first command loads the sample. For the beginning only the second parameter in the first command is relevant - the name of the file we want to play. The rest is important when we want to use more than one sample, play a file from memory, etc. The second command plays the actual sample. The first parameter in the parentheses is the channel number to be used. The second parameter is the handle of the sound to play.

But take care that the file that to be played is not that big! Otherwise, it will take some time until the file is played. This delay occurs because file is first loaded completely into memory.

7.8.1.2 Volume Control for Songs

Now, that the sample is playing and it can also be manipulated it in several ways. Therefore, to make it louder one must use, "FSOUND_SetVolume (handle, 255);" With value 255 the volume is set to the maximum. As with the music, if it had been passed 0(zero) then there would have been silence. The volume can be adjusted for a sample by passing a handle, or adjust a channel by passing the relevant channel number instead of a handle. To pause the sample use, "FSOUND_SetPaused (handle, true);" If there is need to remove the pause. Again the first parameter would be a channel number. "FSOUND_SetPaused (handle, false);" Where as, when there is need to stop the sample then only use, "FSOUND_StopSound (handle);"To clean up the sample use the command, "FSOUND_Sample_Free (handle);"

7.8.1.3 A small simple console example

Nothing fancy here. This snippet just plays the file **sample.mp3**.

```
#include <conio.h>

#include "inc/fmod.h"

FSOUND_SAMPLE* handle;

int main ()

{

  // init FMOD sound system

  FSOUND_Init (44100, 32, 0);
  // load and play sample
  handle=FSOUND_Sample_Load (0,"sample.mp3",0, 0, 0);
  FSOUND_PlaySound (0,handle);
  // wait until the users hits a key to end the app
  while (!_kbhit())
  {
  }
  // clean up
  FSOUND_Sample_Free (handle);
  FSOUND_Close();
}
```

7.8.2 Streams

7.8.2.1 Stream’s Handle

There is need for “FSOUND_STREAM” variable for the handle.

```
handle=FSOUND_Stream_Open("YourFileName",0, 0, 0);
FSOUND_Stream_Play (0,handle);
```

It's much the same as we have the code to play samples so nothing more to say here. But we have to take care that we're using at least version 3.7. In earlier versions the command for opening a stream is different!

7.8.2.2 Volume Control For Streams

To stop a stream we use, “FSOUND_Stream_Stop (handle);” For cleaning up the stream a simple line of code is, “FSOUND_Stream_Close(handle);”

7.8.2.3 A small simple console example

This small example streams the file **sample.mp3**.

```
#include <conio.h>
#include "inc/fmod.h"
FSOUND_STREAM* handle;
void main ()
{
  FSOUND_Init (44100, 32, 0);    //init FMOD sound system
  //load and play sample
  handle=FSOUND_Stream_Open("sample.mp3",0, 0, 0);
  FSOUND_Stream_Play (0,handle);
  //wait until the users hits a key to end the app
  while (!_kbhit())
  {
  }
  FSOUND_Stream_Close(handle);
  FSOUND_Close();   }
```

7.9 Shutting Down The FMOD Engine

To shut down the FMOD sound system one must use, "FSOUND_Close ();"

7.10 Sound Samples Part of Simulation

Mostly use of sample sound effects at the background of the simulation was implemented. Specifically, some of the sample sounds are given in table 7.1.

Table 7.1 Sample Sounds

a.	Helicopter sound samples.
b.	Tank movement and firing.
c.	Arty gun and mortars firing.
d.	Wind/breeze sound samples.
e.	Aircraft sound and missile fire.
f.	Different chirping birds samples.
g.	Sound sample of overall battle scene.
h.	Sound sample of Walking through water
i.	Small Arms fire effects i.e. Infantrymen rifle/machine gun

7.11 Summary

This chapter provides the reality to our simulation as a real battle field. When all above mentioned sound samples were integrated into the simulation then it really looked as if real battle field scenario. Therefore, addition of sounds made the work more presentable and attractive. Use of FMOD API code is very easy and programmer friendly. However, one can improve upon by adding number of other sound effects.

Chapter 8

8. Resources Utilization

8.1 Introduction

To create different models for simulation we have used 3DS in a 3d file format. These can be viewed (using OpenGL) with better performance and real touch/look. This format was used because free libraries were available to view this format in OpenGL. This library was further extended to include multiple model support and individual part colour support. Here are some advantages and disadvantages of using 3DS files.

8.1.1 Merits of using 3DS files

It (3DSMax) is a very popular format, which seems to be quite popular among artists. Unknown parts in the file can be skipped easily. File easy to read. Optional per-vertex texture coordinates.

8.1.2 Demerits of using 3DS files

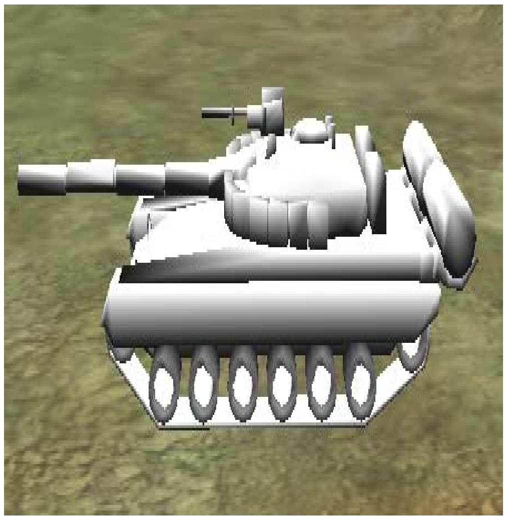
Materials are often stored in separate - unavailable files. Poor normal information stored in the file. Object have a maximum of 2^16 vertices. The 3DS format was chosen because of its popularity. Many 3D objects available on the web are stored in 3DS format. The limitations were overcome by using external tools like Accu Trans.

8.2 Description Of 3DS Models

We used many 3d models in our simulation. Description for some of the models is discussed below.

8.2.1 Tank

Basically we have shown tanks on both side i.e. foxland and blueland. Enemy tanks are depicted in yellow colour scheme where as defender side tanks are shown in white colour scheme. We have depicted a troop of tanks towards defender side and a squadron of tanks towards enemy side. Tanks on enemy side are automated in a sense that when simulation starts, then these tank come in single line formation for advance towards international border for crossing the bridges at nullah. But before move these tanks carry out firing at different locations in the AOR(area of responsibility) of blueland force. Model picked up for both sides is the same having low number of polygons. Figures for both Blueland and Foxland tanks are shown below as fig 8.1and 8.2 respectively.



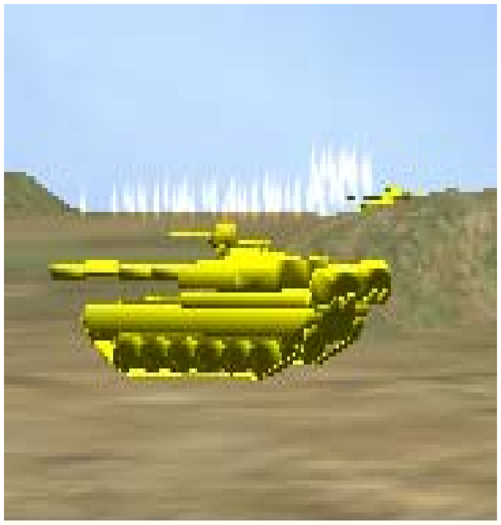


Figure 8.1 Blueland Tank

Figure 8.2 Foxland Tank

8.2.2 Rifle

For the gr cbt, soldier is most important element of war and hence is the rifle as his personal weapon. Large variety of weapons in shape of rifle were available. However, the model of rifle we used here is having very low number of polygons. Figure for rifle is shown below as fig 8.3.

Figure 8.3 Rifle for Infantry Man

8.2.3 Gun/Mortars

It is said that mortars are the arty sp for company commander during any operation of war may it be defence or attk. Hence, we also have depicted some mortars at the defender side behind the DCB bund. Figure for Gun shown below as fig 8.4.

Figure 8.4 Arty Gun

8.2.4 Soldier

Soldier is most important element of war for any gr cbt operation. We have faced lot of difficulty in getting and creating the true model of a soldier. We needed two kinds of soldiers i.e. static and moving. The static soldiers are to be used within defences and moving soldier characters were to be used with attk elements. Figs for Blueland soldier models are shown below as fig 8.5and 8.6 respectively.

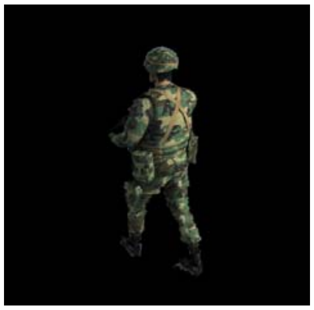


Figure 8.5 Infantry Man (Back)

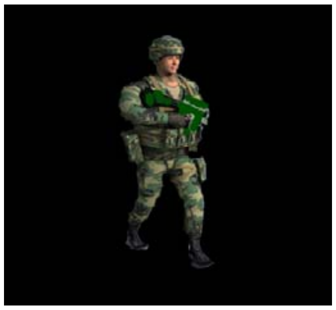


Figure 8.6 Infantry Man (Front)

Figures for Foxland soldier models are shown below as fig 8.7and 8.8 respectively.

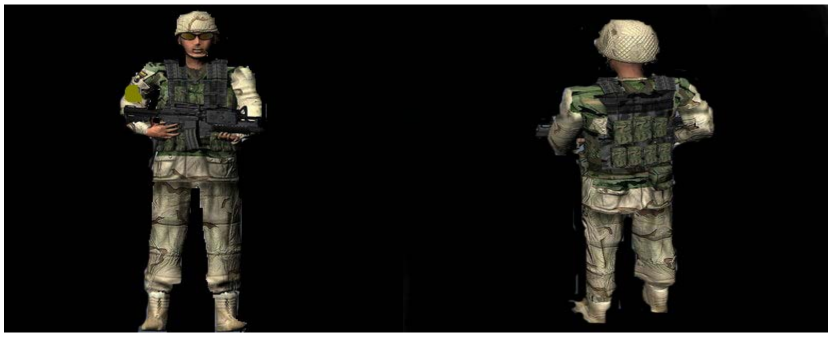


Figure 8.7 Infantry Man (Front)   Figure 8.8 Infantry Man (Back)

8.2.5 Fence And Sign Boards

It is a SOP(Standing Order Procedure) that each country defends its homeland by placing fence all along its border area. We have an international border in between foxland and blueland which is marked with border fence. We also have demarcated the area of mine fields in front of defences by using model of fence. As per international rules the indication for presence of mine fields is also shown with the help of sign boards. Figure for the fence and sign board are shown below as fig 8.9 and 8.10 respectively.

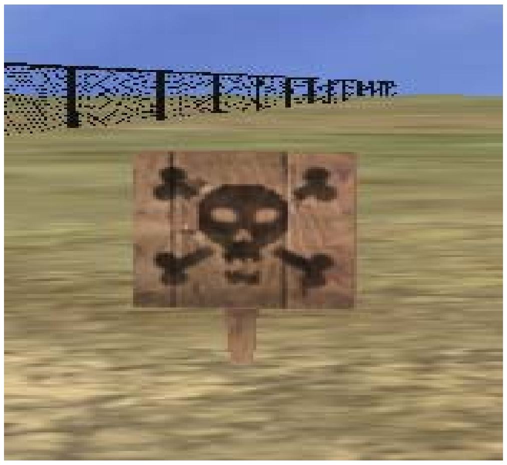


Figure 8.9 Sign Board

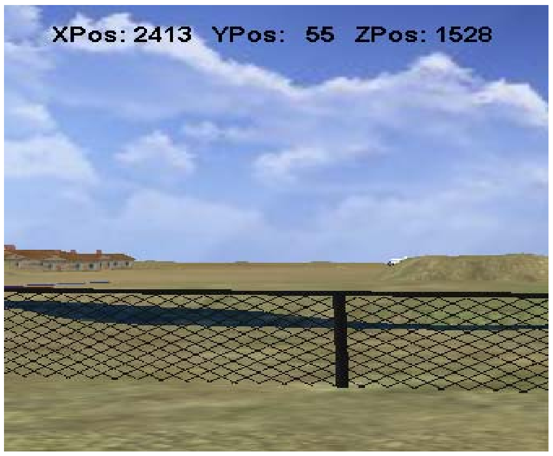


Figure 8.10 Fence

8.2.6 Bridges

Within our area of operation shown in simulation we have nullah and a water channel as part of DCB. Therefore it was imperative to construct the bridges over them for the crossing purpose. At the DCB we have placed a bridge model with low number of polygons. Where as, at nullah we have placed three bridges at different locations. These bridges are the textures mapped with the terrain. Shown as Figure 8.11 and 8.12.

Figure 8.11 Textured Bridge

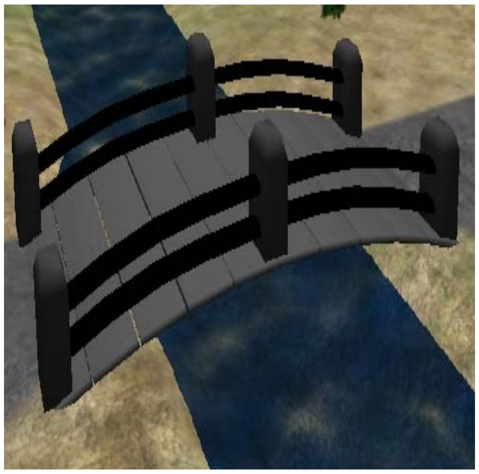


Figure 8.12 Bridge At DCB

8.2.7 Aircraft

When we described the information about enemy in chapter 2 then at that time we also mentioned that enemy enjoys air superiority and he is likely to launch air attk at the start of hostilities. To depict this situation we have used a model of bomber aircraft. Three instances of this model are generated. On pressing specific key of keyboard enemy air attk is launched and from defended locality we can view that aircrafts are heading towards the DCB in extended formation. They carry out bombardment with missiles in general area of DCB and then fall back in the direction from which they attack. Shown as figure 8.13.

Figure 8.13 Bomber aircraft

8.2.8 Missile

During air attk the enemy aircrafts fire the missiles at defended locality. Normally in operations of war we see that the missiles are not visible clearly due to its speed, so same factor applies here. Only very short blink of missile appears and we witness the fire effects where missiles are hit. Missile fire is shown using Bezier curve equation.

8.2.9 Helicopter

We have also shown the helicopter at the side of defender. This helicopter is just used for recce purposes. Model of this helicopter is made up of low polygons. Figure for the Modelled helicopter is shown below as fig 8.14.

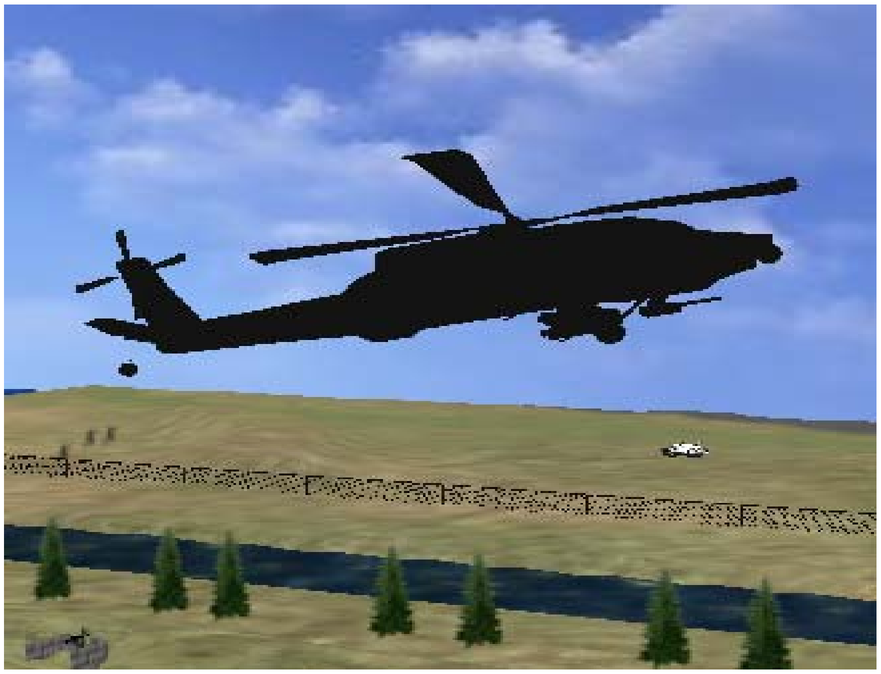


Figure 8.14 Helicopter From Blueland

8.3 Accu Trans 3D As A Tool

Accu Trans provides accurate translation of 3D geometry between the file formats used by many popular modeling programs. Positional and rotational information for the 3D meshes is maintained. Also many materials attributes, such as color, index of refraction, reflection, secularity and Phong shading, are transferred between the files. Accu Trans can successfully translate between many file formats. Some of the file formats are mentioned in table 8.1 on next page.

Table-8.1 Accu Trans Import / Export File Formats

File Format	File Extension	Read	Write
3D Metafile	.3dmf	No	Yes
3D Studio	.3ds, .asc, .prj	Yes	Yes
AutoCAD DXF	.dxf	Yes	Yes
DirectX	.x (ASCII & Binary)	No	Yes
Imagine (Original & New Formats)	.iob	Yes	Yes
Turbo Silver (Amiga)	.ts	Yes	Yes
LightWave(LWOB and LWO2 Formats)	.lwo	Yes	Yes
Lightscape	.lp	Yes	Yes
Maya	.ma	No	Yes
Maya	.rtg	Yes	No
POV-Ray 3.0	.pov	No	Yes
RealiMation Version 4.1	.rbs	Yes	Yes
RenderWare	.rwx (ASCII only)	Yes	Yes
Sculpt (Amiga)	.scene	Yes	Yes
Softimage | XSI	.xsi (ASCII - version 3.5)	No	Yes
StereoLithography	.stl (ASCII & Binary)	Yes	Yes
trueSpace	.coa, .cob	Yes	Yes
VideoScape (Amiga)	.geo	Yes	Yes
Viewpoint Scene	.mtx	No	Yes
VRML 1.0 & 2.0	.wrl (ASCII only)	Yes	Yes
Wavefront	.obj	Yes	Yes
X3D	.x3d	No	Yes
XGL, ZGL	.xgl, .zgl	Yes	Yes
XYZ	.xyz (ASCII and Binary)	Yes	No

When a file is read, objects with more than one material / colour assigned are divided into sub objects. Convert coplanar triangular faces into Quads (4 sided polygons).

8.4 Terrain Features

It is very important element of gr cbt simulation because without the terrain all is of no use. There are several ways and methods of terrain generation. We will be discussing only two of them. However, for our terrain generation we have used the idea of height maps, where we created .raw files.

8.4.1 TerraGen

It is a non real time terrain generator. In this is a software developed by a single individual and detailed scenery is generated using fractal algorithms. Height maps for the project can be imported using its terrain dialog.

8.4.1.1 Terrain Dialog

The Terrain Dialog is the usually the first part of TerraGen that is encountered. Here the terrain can be generated (or open existing ones) or modified it in various ways. The surface map can also be created.

8.4.1.2 Import/Export of Terrains

The terrain can be imported / exported as a raw binary file (7 bit per pixel) of resolution 257*257. The terrain can also be exported in VistaPro-compatible binary format and LightWave 3D Object (LWO) files. The RAW option is very useful as it allows creating grey scale images in a normal paint program and importing them as landscapes. There are utilities that enable to convert USGS DEM (digital elevation map) files for import into TerraGen to allow rendering real-world scenes .The button just below the terrain view displays the width of the terrain in meters. Click on it to change the scale used, or to change the resolution of the terrain. However, we are not using this method (TerraGen) for generating terrain.

8.4.2 Height Map

The terrain was created terrain using photo shop for making raw height map files. A height map is a set of Gray scale colour values that determines the "height." Here a height map from a .raw file was read. A texture was applied over the entire terrain. The terrain is rendered using triangle strips. Tiling a second texture on top of the first one, for giving the appearance in more detail. All above is called detail texturing, which can add a great deal of realism to a scene. Multi-texturing is used to achieve this neat effect.

First of all the height map is read from the .raw file. There is no need to pass in the same vertex more than once. Each 2 vertices are connected to the next 2. There was a need to reverse the order every other column. If not done so, one will get polygons stretching across the whole terrain. Multi texturing was added for a detailed texture application over terrain. Normal Multi texturing function was used, and properties were changed. Work on the texture matrix was also done. Instead of calculating the (u, v) coordinates for the tiled detail texture, it was just assigned the same (u, v) coordinates as the terrain texture. Then scaled the texture coordinates by entering the texture matrix mode and applying scale value.

8.5 Trees

Trees are part of terrain and were catered during terrain generation but, loaded the separate model of trees on the ground. Blended/masked models of trees are in figure 8.15.

Figure 8.15 Tree shown as Blended And Masked

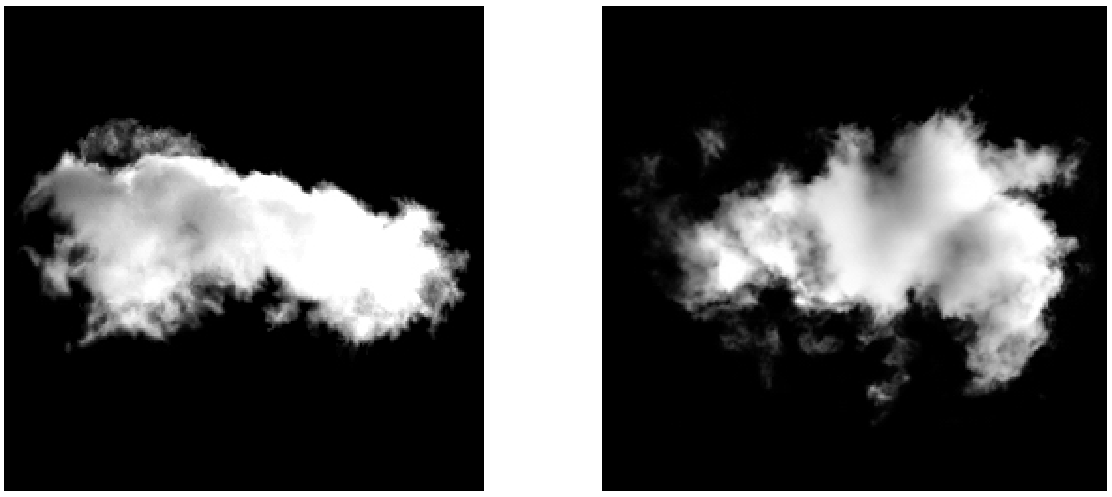


Figure-8.16 Sample Clouds

8.7 Villages And Buildings

To give real touch, some buildings and villages on both sides of the border are depicted. Initially, it was not giving real touch. So option available to use were “textures”.

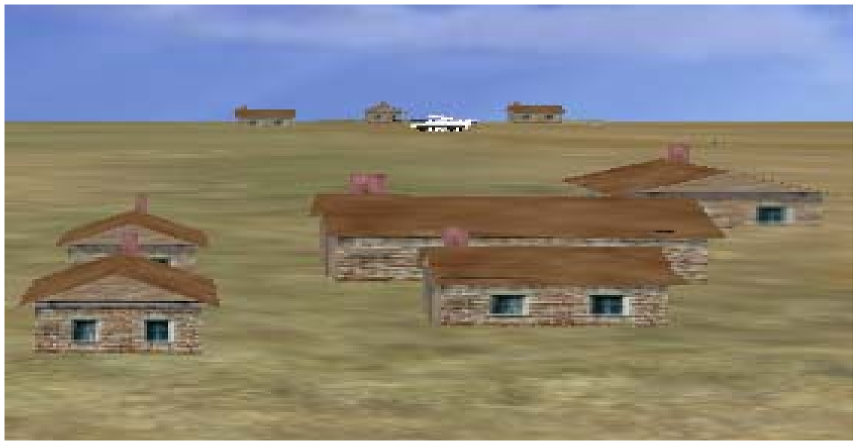


Figure 8.17 Villages/Buildings

8.8 Bunkers

The bunkers were made on same lines the way the houses were created. These bunkerswere placed on DCB. One bunker is shown as fig 8.18.

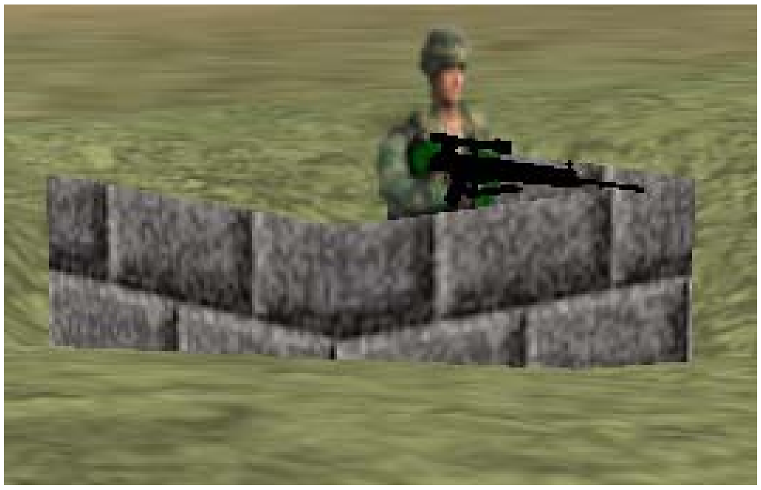


Figure 8.18 Bunker with soldier

8.9 Terrain Elements

I have mentioned that terrain is built in the photo shop as .raw file. We have made four raw files for our terrain generation. One raw file has a size of 3000 * 3000 pixels. So our terrain has total size of 6000 * 6000 pixels. We have used grey scale values to depict elements of the terrain. Some of the elements are: Plain ground, Ridges of medium height, Kidney bunds (small ridges in the shape of kidney), Roads and tracks and Water Bodies.

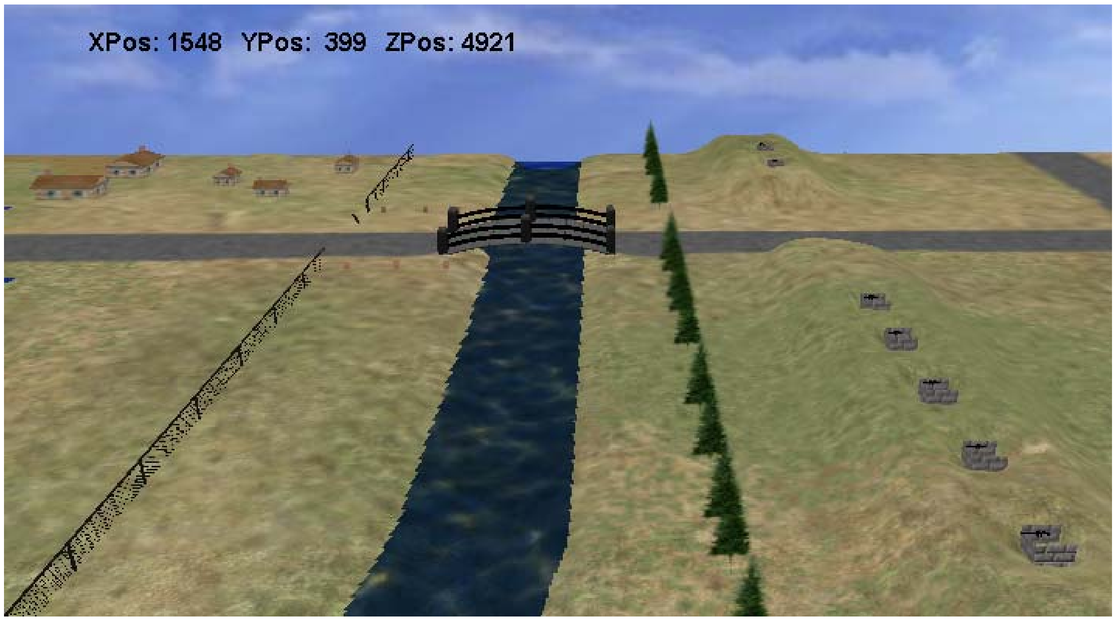


Figure 8.19 General View Of Terrain

8.10 Summary

This chapter is most important with a viev that it provides us the complete picture that how all the resources were created and utilized in an efficient manner to produce a simulation of its own kind for the first time in the history of Mil College of Signals and Pak Army. Detailed description for each model is also shown along with figures.

Chapter 9

9. Menu System for Simulation

9.1 Introduction

The operation of the individual modules/models described in appropriate chapters. Whereas, this chapter describes the menu Design used to conduct the simulation. Basically we require menu so that our simulation could be user friendly. Through these menus we can view particular area of terrain, move through whole terrain i.e. use of camera controls, can manipulate the blueland forces tanks, can start the simulation, automate the enemy tanks, launch an air attk and carrying out the fire of missiles, tanks and rifles.

9.2 Why Menu is Important?

Menus are computer programs. They guide the user step by step through procedures to be followed. They enable the Tanks to firing readiness. The communication between the user and the computer takes place using the display and the keypad. It makes the simulation user friendly. It becomes easy to understand that how the program actually works.

9.3 Menu Design Tool

Lot of choices were available to us to implement the menu design, like use of different available API’s. But for the reason of simplicity and friendliness we have used “***FONTS”*** to implement our menu design. There are number of Fonts options are available to be used. [10]

9.3.1 Choice Among Fonts

Variety of fonts types are available, which can be implemented in OpenGL. Some of the fonts are: Bitmap Fonts, Outline Fonts and Texture Fonts. However, the "***BITMAP FONTS***" are used for menu design. Reason for this adoption was that the bitmap fonts give 100% better look then texture fonts and they are easy to implement as well. [10]

9.4 Keypad Operation

9.4.1 Start of simulation

When file is run then the menu system screen appears. It has x, y and z positions of camera at the top left corner of screen. Press **'S'** to start the simulation.

9.4.2 Camera Controls

From keyboard one can Yaw, Pitch, Roll, Move right/left, up/down and strafe right/left. From mouse one can change the direction of camera. Where ever the arrow head is moved the camera eye is pointed towards that direction.

9.4.3 Tank Controls(Blueland)

The tks are depicted in white color and one can control them by the key **'B'.** Moreover, the control of one blueland's tank is with user as shown in table 91.

Table 9.1 Tank Movement Controls

a.	Move Forward of Tank	**'W'**
b.	Move Backward of Tank	**'X'**
c.	Turning Right of Tank	**'D'**
d.	Turning Left of Tank	**'A'**

9.4.3 Tank Controls and Air Attk (Foxland)

Enemy tanks have yellow color scheme and they are controlled with the key **‘F’.** Air attk of three bomber aircrafts is shown, Which are depicted as approaching from foxland area and heading towards DCB. Along with this these aircrafts also carry out the bombing of the missiles in general area of DCB. We have allocated the key **‘J’** for this all action to be displayed.

9.4.4 View Specific Location

As our area of interest is very large so to heve a view of specific are we had to move the camera manually to that location. To overcome this problem different keys were selected to reach at appropriate location within no time by just pressing single key. These key conrols are shown in table 9.2.

Table 9.2 Controls for Viewing Specific location

a.	Enemy tanks fmn 1(Left)	**‘F1’**
b.	Enemy tanks fmn 2(Center)	**‘F2’**
c.	Enemy tanks fmn 3(Right)	**‘F3’**
d.	Enemy Area View(Center)	**‘F6’**
e.	Marshy area and clump of trees	**‘F7’**
f.	Move to DCB at Left corner	**‘F8’**
g.	Move to DCB at central bridge	**‘F9’**
h.	Move Standing patrol area	**‘F11’**

9.4.5 Tanks Firing

Tanks firing for blueland and foxland is being controlled by the same keys from which the tanks are moved.

9.4.6 Rifle Fire

We have rifles in the bunkers and these are divided into groups for the purpose of showing their fire effect Total of 36 bunkers on DCB, they are divided into 4 groups of 9 weapons in each group. Keys allocated to start and stop the fire are shown in table 9.3.

Table 9.3 Rifle Fire Controls

a.	Left gp	**‘7’**
b.	Centre-Left gp	**‘8’**
c.	Centre-Right gp	**‘9’**
d.	Right gp	**‘0’**

9.4.7 Guns Fire

Six arty guns were depicted and these are placed behind the DCB. Calculated fire effect of these guns was shown and allocated numeric keys to control the fire of each gp are shown in table 9.4 and 9.5 respectively.

Table 9.4 Left Portion of DCB

a.	Left gp	**‘1’**
b.	Center gp	**‘2’**
c.	Right gp	**‘3’**

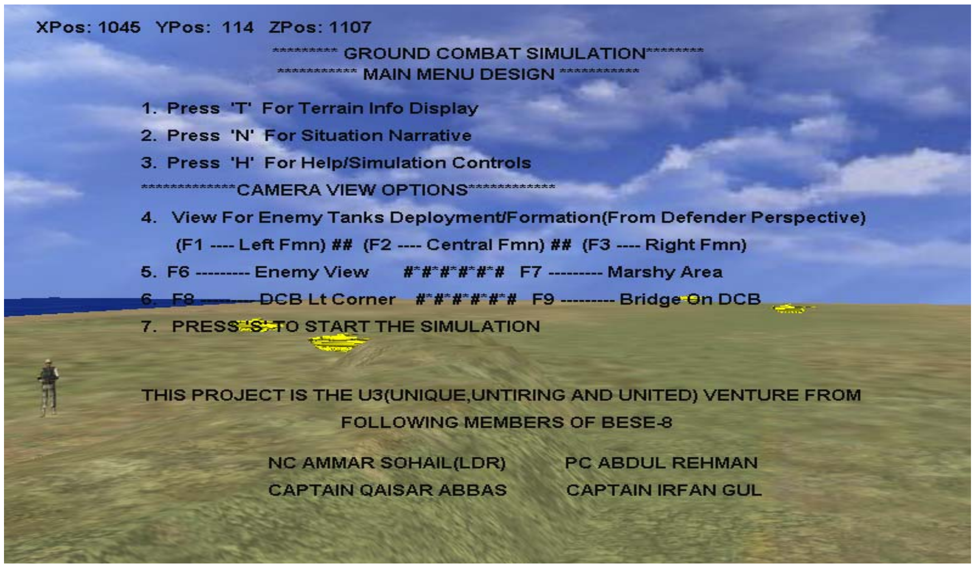
Table 9.5 Right Portion of DCB

a.	Left gp	**‘4’**
b.	Center gp	**‘5’**
c.	Right gp	**‘6’**

9.5 Main Menu Form

As the first appearance we get the main form. We have the options to move to any sub form and fall back to main Formby pressing the key **‘E’**. Moreover, we dynamically designed this menu so that even if we want to visit any of the form during simulation then we can just see the details b pressing the respective keys. Sub forms in the menu are: Terrain Information, Help screen, Battle situation.

Main form and sub forms are shown as figure 9.1,9.2,9.3 and 9.4 respectively.



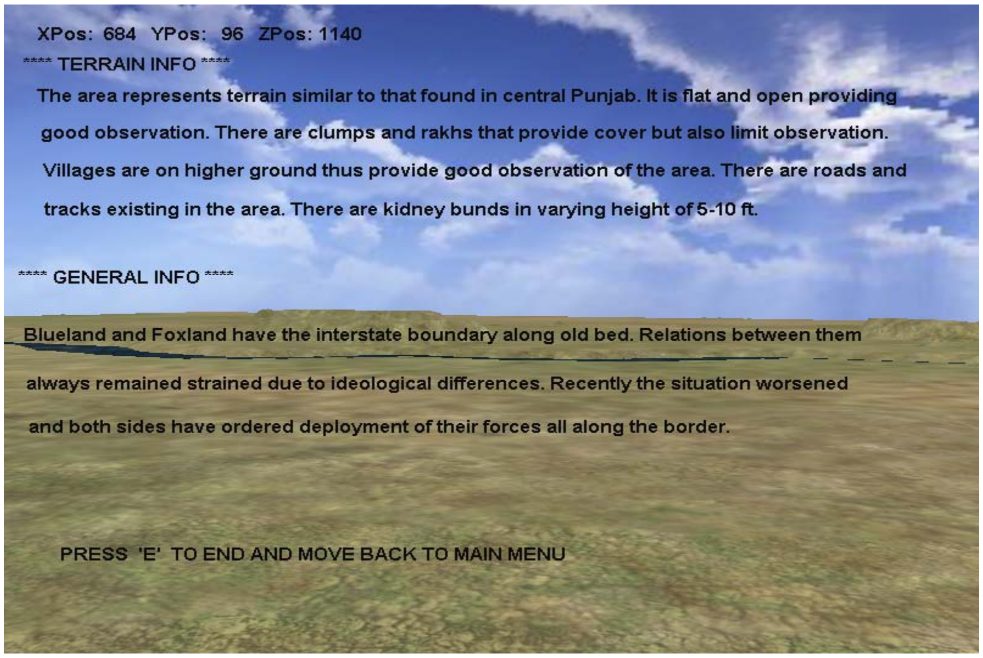
```
XPos: 1045   YPos:  114   ZPos: 1107
                    ********* GROUND COMBAT SIMULATION*********
                    *********** MAIN MENU DESIGN ***********

        1.  Press  'T'  For Terrain Info Display
        2.  Press  'N'  For Situation Narrative
        3.  Press  'H'  For Help/Simulation Controls
        **************CAMERA VIEW OPTIONS*************
        4.   View For Enemy Tanks Deployment/Formation(From Defender Perspective)
             (F1 ---- Left Fmn) ##  (F2 ---- Central Fmn) ##  (F3 ---- Right Fmn)
        5.  F6 --------- Enemy View      #*#*#*#*#*#   F7 --------- Marshy Area
        6.  F8 --------- DCB Lt Corner    #*#*#*#*#*#   F9 --------- Bridge On DCB
        7.   PRESS 'S' TO START THE SIMULATION

        THIS PROJECT IS THE U3(UNIQUE,UNTIRING AND UNITED) VENTURE FROM
                          FOLLOWING MEMBERS OF BESE-8

                    NC AMMAR SOHAIL(LDR)          PC ABDUL REHMAN
                    CAPTAIN QAISAR ABBAS          CAPTAIN IRFAN GUL
```

Figure 9.1 Main Menu Form



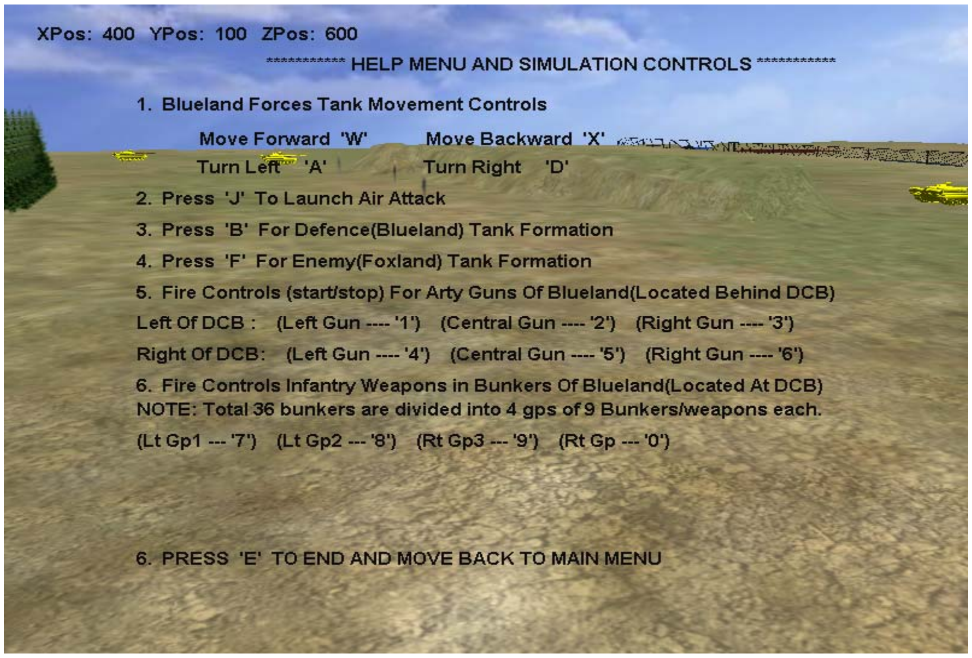
```
XPos: 684  YPos: 96  ZPos: 1140
**** TERRAIN INFO ****
The area represents terrain similar to that found in central Punjab. It is flat and open providing
good observation. There are clumps and rakhs that provide cover but also limit observation.
Villages are on higher ground thus provide good observation of the area. There are roads and
tracks existing in the area. There are kidney bunds in varying height of 5-10 ft.

**** GENERAL INFO ****

Blueland and Foxland have the interstate boundary along old bed. Relations between them
always remained strained due to ideological differences. Recently the situation worsened
and both sides have ordered deployment of their forces all along the border.

PRESS 'E' TO END AND MOVE BACK TO MAIN MENU
```

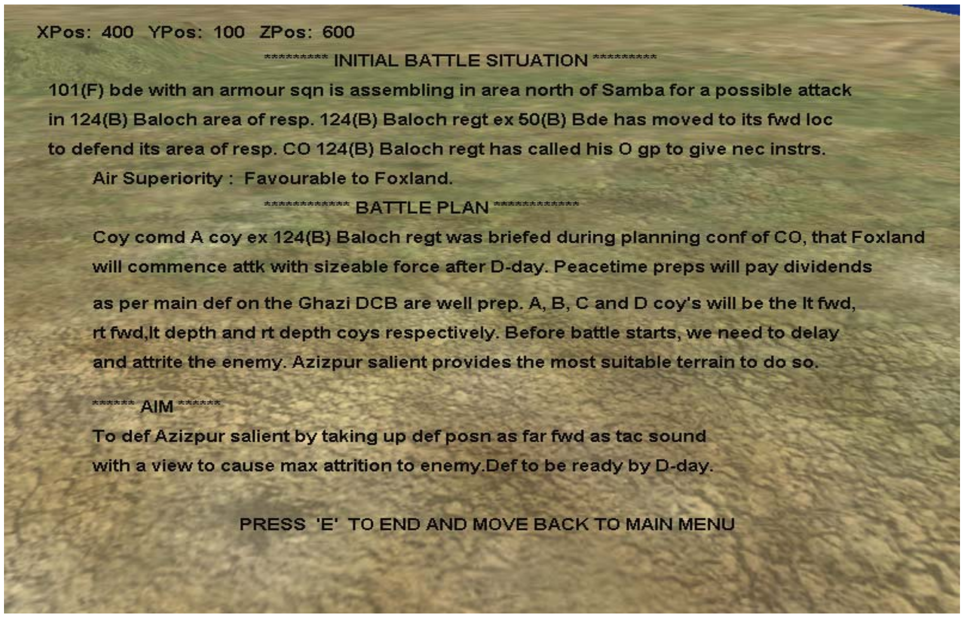
Figure 9.2 Terrain Information Form



```
XPos: 400  YPos: 100  ZPos: 600
*********** HELP MENU AND SIMULATION CONTROLS ***********
1. Blueland Forces Tank Movement Controls
      Move Forward 'W'        Move Backward 'X'
      Turn Left    'A'        Turn Right    'D'
2. Press 'J' To Launch Air Attack
3. Press 'B' For Defence(Blueland) Tank Formation
4. Press 'F' For Enemy(Foxland) Tank Formation
5. Fire Controls (start/stop) For Arty Guns Of Blueland(Located Behind DCB)
Left Of DCB :   (Left Gun ---- '1')  (Central Gun ---- '2')  (Right Gun ---- '3')
Right Of DCB:   (Left Gun ---- '4')  (Central Gun ---- '5')  (Right Gun ---- '6')
6. Fire Controls Infantry Weapons in Bunkers Of Blueland(Located At DCB)
NOTE: Total 36 bunkers are divided into 4 gps of 9 Bunkers/weapons each.
(Lt Gp1 --- '7')  (Lt Gp2 --- '8')  (Rt Gp3 --- '9')  (Rt Gp --- '0')


6. PRESS 'E' TO END AND MOVE BACK TO MAIN MENU
```

Figure 9.3 Help Screen Form



```
XPos: 400  YPos: 100  ZPos: 600
                              ********* INITIAL BATTLE SITUATION *********
 101(F) bde with an armour sqn is assembling in area north of Samba for a possible attack
 in 124(B) Baloch area of resp. 124(B) Baloch regt ex 50(B) Bde has moved to its fwd loc
 to defend its area of resp. CO 124(B) Baloch regt has called his O gp to give nec instrs.
      Air Superiority :  Favourable to Foxland.
                              ************ BATTLE PLAN ************
      Coy comd A coy ex 124(B) Baloch regt was briefed during planning conf of CO, that Foxland
      will commence attk with sizeable force after D-day. Peacetime preps will pay dividends
      as per main def on the Ghazi DCB are well prep. A, B, C and D coy's will be the lt fwd,
      rt fwd,lt depth and rt depth coys respectively. Before battle starts, we need to delay
      and attrite the enemy. Azizpur salient provides the most suitable terrain to do so.

      ****** AIM ******
      To def Azizpur salient by taking up def posn as far fwd as tac sound
      with a view to cause max attrition to enemy.Def to be ready by D-day.

                           PRESS  'E'  TO END AND MOVE BACK TO MAIN MENU
```

Figure 9.4 Battle Situation Form

9.6 Summary

In this chapter we have worked on the GUI which is being shown in shape of main menu form and other sub forms. All this is done using the Bitmap Fonts of the OpenGL library. We have depicted various simulation controls along with the brief view of the terrain information and battle situation.

Chapter 10

10. HLA – High Level Architecture

10.1 Introduction

This chapter tries to provide a lot of information regarding HLA (High Level Architecture) in order to advance the programmer unfamiliar to HLA. It is a novel way to build military simulation systems. HLA is not an area that one can learn in a day. HLA involves climbing a high learning curve, however it is worthwhile. HLA is an impressive piece of work. HLA features many fine properties, such as dynamic discover and delete functionality of the sub-systems participating in the simulation and dynamic ownership handling. The major aim of HLA is to standardize the interconnection between simulators, hence provide easy ways of connecting simulators running HLA. In December 1997, HLA was accepted as a draft IEEE standard. RTI versions 1.0.* are government-controlled development. RTI 2.0 will be industry developed, using an open competitive design process. Currently version 1.0.3 is out for the platforms in table 10.1:

Table 10.1 HLA Platforms

a.	Silicon Graphics (IRIX > 6.2)
b.	Sun Sparc (Solarix > 2.5+)
c.	Hewlett Packard (HP-UX > 10.20)
d.	IBM (AIX > 4.1.5)
e.	DEC Alpha (OSF1 > V4.0)
f.	Windows (95/NT 4.0)

Version 1.3 should have implemented all announced services, however version 1.0.3 is stable and equipped with functions sufficient for most common simulators. [1]

10.2 HLA Terminologies

First some words about HLA terminologies. HLA is not only a simulator architecture. It contains specifications about, how to operate simulators together; how and what data to communicate, specifications about the individual simulators, what services should be offered by a simulator and how the simulator works, a skeleton for designing HLA compliant simulators and a test and verification scheme. [1]

10.2.1 Runtime Infrastructure (RTI)

The general purpose distributed operating system software which provides the common interface services during the runtime of an HLA simulation. The RTI can be conceived as a post office for all data traffic in the simulator. Note that all information that change during a simulation, that needs to be shared between several simulators needs to pass through the RTI.

10.2.2 Object Model Template (OMT)

Template used to describe an object. For HLA specification a tool, called OMDT, which can be downloaded from the HLA homepage. The OMDT which enforces specification of objects in the OMT framework. This tool is currently only available for Windows NT.

10.2.3 Federation Object Model (FOM)

An identification of the essential classes of objects, object attributes, and object interactions. The FOM does not contain information about the actual objects in the simulation (the federates), but only about the possible object classes in the simulation.

10.2.4 Simulation Object Model (SOM)

A specification of the intrinsic capabilities that a simulator1 offers to federations. In each SOM a set of attributes (parameters) are listed by type, cardinality, units and specification about how they are updated. The SOM can be compared with a class description, hence several federates can share a given SOM, if they are alike in specification, e.g. all F16 fighter simulators could have the same SOM.

10.2.5 Federate and Federation

It is a member of a HLA Federation. A federate is an actual simulator with capabilities specified by its SOM. Note that several federates can have the same SOM.

Where as federation is a named set of actual interacting federates (all actual simulators), a common federation object model (the FED file), and supporting RTI, that are used as a whole to achieve some specific objective.

10.2.6 Class and Attribute

Each federate is part of a class, which is the SOM. The C++ concept of a Class maps almost perfectly onto the SOM. Where as, an attribute is a parameter with one or several values in a SOM, that can be changed during a simulation. Other federates may or may not desire to get changes of the attribute depending on the actual simulator settings. The federate who offers the attributes to others will **publish** an attribute and the federate that needs the attribute changes will **subscribe** to the attribute.

10.2.7 Interaction

Ordinarily the fundamental way data is communicated from one simulator to another is via changes in attributes. The other supported way is via interactions. The intended usage is for orders, i.e. data communication that is only relevant to that very instance in time. The interaction is in principle not stored, for later retrieval. It is possible to use interaction for many things, such as offering a service close to function calls in C++. In Figure 5.1 is shown a typical HLA simulation. It is seen that several federates are running together with two standard support utilities: The **rtiexec** is a daemon program that is started on the server machine. It supports any HLA simulation on the machine including disjoint simulators (e.g. combat and industrial without interconnection). The **fedex** is a process that supports a given federation. It is started automatically by the rtiexec when the first federate joins the federation (a shell will pop up displaying the actual federates).

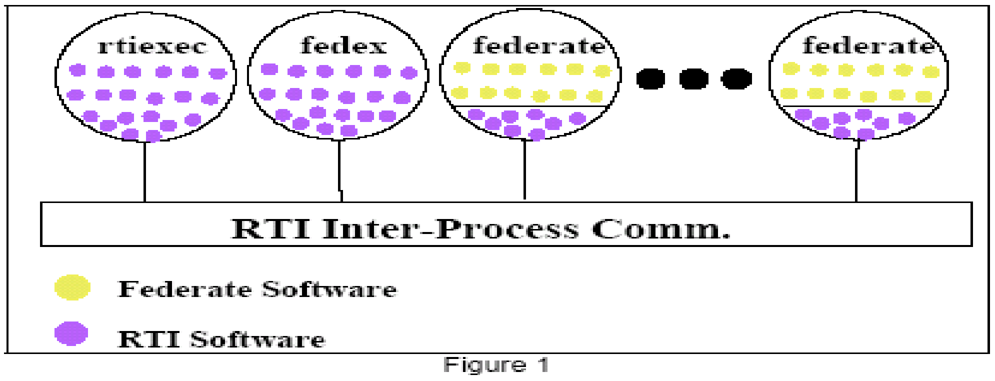


Figure 10.1 RTI Structure

10.3 The Ten Basic HLA Rules

HLA is based on five rules for the federation and five rules for the federates. [1]

5.3.1 Five rules for federations

1. Federations shall have an HLA federation object model (FOM). *A FED-file will contain the FOM information necessary.*

2. In a federation, all object instance representation shall be in federates, not in RTI. *Data cannot be stored in the RTI. The RTI will only route data.*

3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI. *It is highly illegal to send data that changes over time directly between simulators. This rule is made in order to ensure stability and consistency of the simulation.*

4. During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification. *All interactions must follow the HLA specifications.*

5. During a federation execution, an attribute of an instance of an object shall be owned by at most one federate at any time. *Ownership handling is also supported in HLA and an object (a simulator) can at most be owned by one simulator.*

10.3.2 Five rules for federates

1. Federates shall have an HLA simulation object model (SOM). *The OMDT will make this documentation showing all attributes, their types and sizes.*

2. Federates shall be able to update and/or reflect any attributes of objects in their SOMs and send and/or receive SOM interactions externally, as specified in their SOMs. *The SOM must specify that attributes will be updated given certain conditions, e.g. a defense system may be initiated only if an enemy is within a certain distance.*

3. Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOMs. *The SOM should document the conditions of ownership handling.*

4. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes of objects, as specified in their SOMs. *In version 1.3 of the*

RTI software it is possible to provide attribute changes if certain logical function is fulfilled. For version 1.0.3 this is yet to come.

5. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation. *Time control is of course very important. HLA offers several ways that can be mixed. The important issue is that time in a simulator must be consistent, i.e. have a increasing flow.*

10.4 HLA infrastructure

10.4.1 The High Level Architecture Run-Time Infrastructure Programmer's Guide.

RTI.rid (found in **$RTI_CONFIG**) is a file that is a global file and used primarily by the **rtiexec** to determine which RTI server machine to use. For the server machine uses the line RTI_EXEC_HOST local host and any client machine should change the line to RTI_EXEC_HOST my_rti_servermachine where my_rti_servermachine should be set to the name of the actual server machine. In the same file, the port used for the RTI is set (RTI_EXEC_PORT). The default is 18134. Besides the RTI_EXEC_HOST all other parameters are ordinarily left unchanged. [1]

It should be noted that the FED-file, which contains the FOM information, can be generated by the OMDT tool. In the next picture is shown a screen dump showing the OMDT tool on a NT machine (only available for NT). It is quite easy to design the classes, the attributes including their types etc., and then the FED-file can be stored directly using a save-as operation. The OMDT can verify that the model is consistent and well documented, but it cannot generate actual code. However it is a good for defining a common data model and to guide the implementation of the HLA code. The OMDT has rather easy documentation.

10.4.2 The first example of an HLA simulator

If one was to construct a simple HLA simulator, which should illustrate the core of HLA, one example could be a Driver and Policeman federation. A number of drivers is running at individual speeds and a federate of class Policeman will monitor the speed of the drivers and compare them to a speed limit. If they speed the Police Man will send an interaction to the driver to stop. The Driver should then subscribe to the Speed Limit and maybe also to the Position of the Policeman. The implications are rather obvious. Initially a FED-file must be made, the FED-file, primarily used by the **fedex**. All federates should agree on the SOM and attribute names, hence the FED-file to be copied to the **$RTI_CONFIG** directory on all participating machines. For a simple two class example the file, here called **driver.fed** could look like

```
;; ----------------------------------------------------------
;; Federation Execution Data (FED)
;; ----------------------------------------------------------
(fed
;; ----------------------------------------------------------
;; object, class, and attribute definitions follow
;; ----------------------------------------------------------
(objects
(class Driver
(attribute Name FED_RELIABLE FED_RECEIVE)
(attribute Speed FED_RELIABLE FED_RECEIVE))
(class Policeman
(attribute Name FED_RELIABLE FED_RECEIVE)
(attribute Position FED_BEST_EFFORT FED_RECEIVE)
(attribute SpeedLimit FED_RELIABLE FED_RECEIVE)))
;; --------------------------------------------------------
;; interactions, class, and parameter definitions follow
;; --------------------------------------------------------
(interactions
(class Communication FED_RELIABLE FED_TIMESTAMP
(parameter InterAction))))
```

The idea is that for each object class (SOM) the name of the Class (Driver and Policeman respectively,and each of the attribute names are listed. The same applies for the interaction classes. Note that the FED_RELIABLE means that the attribute changes MUST be send to the federates that desire this . On the other hand if an attribute is not critical but should most of the time be transmitted, then FED_BEST_EFFORT can assist the RTI in the usage of time. If the RTI has to pass a huge amount of data, then these “best effort” attributes can be dumped.

Assume that federate A and B is of class Driver and federate C is of class Policeman. The Policeman federate subscribes to all Speed attributes of the Driver class and he will get the speed of A and B, whenever changed. If Speed exceeds SpeedLimit then C will send an interaction InterAction of Interaction class Communication to the driver. All federates should subscribe to this interaction class. This is indicated in the following diagram.

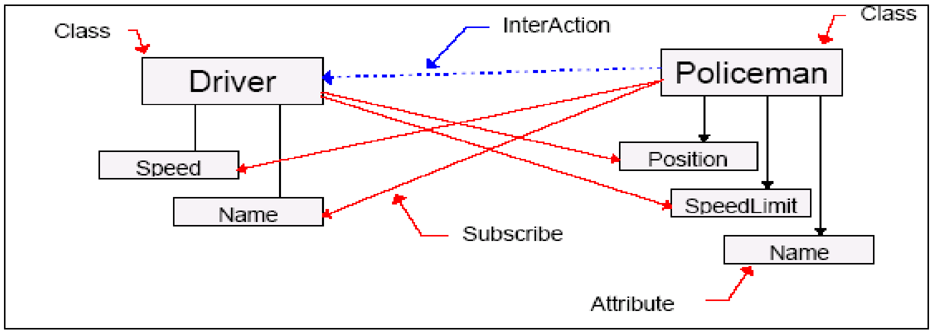
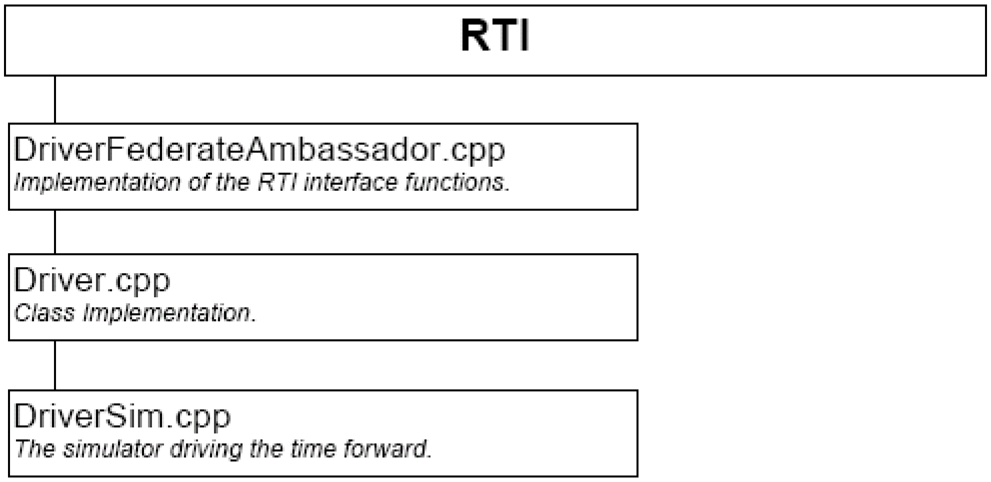


Figure 10.2 Interaction Class

10.5 C++ structure of a simulator

In the Driver/Policeman example the implementation is often split in three parts:

- A federate ambassador which contains a fixed set of functions that can be called from the RTI.
- The class implementation of the internal structure that defines the class.
- A small simulator, which primarily initializes the simulator and controls the time advancements.


RTI
DriverFederateAmbassador.cpp
Implementation of the RTI interface functions.
Driver.cpp
Class Implementation.
DriverSim.cpp
The simulator driving the time forward.

10.5.1 The Federate Ambassador

The federate ambassador contains a fixed set of virtual C++ functions, that are called from the RTI.

discoverObject When an object is initialized and recognized by the RTI, then the RTI will signal the other subscribing federates, that the object if online.

removeObject Messages that a specific federate now has left the federation.

startUpdates Orders the federate to start delivering attribute values whenever changed. The information is specified at class and attribute level.

stopUpdates Orders the federate to stop delivering attribute values. The information is specified at class and attribute level.

reflectAttributeValues Called by the RTI with a set of new attribute values from other federates, that was subscribed to. This function will not normally call an Update function in the relevant federate class.

provideAttributeValueUpdate Called by the RTI instructing the federate to deliver the value of a given attribute.

startInteractionGeneration Orders the federate that other federates are ready to receive specified interactions.

stopInteractionGeneration Orders the federate that none of the other federates will receive specified interactions.

receiveInteraction Called by the RTI with a set of new interactions.

10.5.2 The core of the Driver class

```
class Driver
{
```

```
public:
Driver();
Driver( RTI::ObjectID id);
Driver( int ID);
~Driver();
static Driver* Find( RTI::ObjectID objectId );
static void Init( RTI::RTIambassador* rtiAmb );
void PublishAndSubscribe();
void Update(RTI::FederationTime& newTime );
void Update(RTI::AttributeHandleValuePairSet& theAttributes );
void Update(RTI::InteractionClassHandle theInteraction,
RTI::ParameterHandleValuePairSet& theParameters );
void SetSpeed( const double & );
void SetName( const char* );
private:
char * driver_Name;
double driver_Speed;};
```

Besides a set of constructors and a destructor and functions to set attributes (SetSpeed and SetName) the three Update functions are the core of the HLA class.

- Update(RTI::FederationTime&)

  The function is called by the simulator control (DriverSim) when the time is advanced to the specified time. This function should then integrate and update the internal variables to that time.

- Update(RTI::AttributeHandleValuePairSet&)

  The function is called by the federate ambassador with a set of new attribute values. The function should then update the internal set of variables according the attribute values. Some of which will be directly copied to internal variables, while others might indirectly imply changes.

- Update(RTI::InteractionClassHandle,RTI::ParameterHandleValuePair Set&)

  This function is the analog of the previous one, but for interactions. The function will update internal variables. The concept of interactions (compared to attribute changes) intended for messages that only have relevance at the given point in time.

10.5.3 Subscription to attributes

All communication with the RTI uses type ID's, which is unique number for each of the class attributes found in the SOM specifications.

This is normally made in the Init function which is a part of the class.

10.5.4 Update an attribute from others

One of the most central parts of an HLA simulator is the Update function that receives new attribute values.

10.6 Main loop of a simulator

The main time loop of the simulator is controlled by a small loop, that requests for a signal to allow update of the current time with a given time step. During the time the simulator spends in the rtiAmb.tick() function the simulator will have calls from the RTI with updates of attributes and interactions. When the signal to allow the a time increment is obtained then this simulator will merely print the time and the speed of the driver.

Subsequently the simulator will call the last of the three Update functions to actually update the state of the federate.

10.6.1 Simulator time control

HLA offers several ways to update the time. Any federate can or cannot be **TimeConstrained** and can or cannot be a part of the **TimeRegulation**. A TimeConstrained federate cannot advance its internal time until RTI allows it. This is used for simulators that should advance time almost simultaneously. Federates that are not TimeConstrained are always allowed to increment time, and changes due to new attributes are as fast as possible. Federates that are a part of the TimeRegulation will block TimeConstrained federates until they too allow time increments to the same time point. The concept of time Regulation sets whether a federate should be also determine when the time is advanced. When *all* time regulating federates desire to advance the time, then they get the go signal.

10.7 Summary

In this chapter we have highlighted the feature which were to be implemented in simulation with regards to HLA compliance. Almost 25-30% of work is complete and there are only few bugs due to the differences in the language/platform implementation of both simulation as well as RTI. The interface classes if JNI are worked out but still we have lot of efforts required in this field to come up with good interface in between simulation(VC++ based) and RTI (Java based).

Chapter 11

11. Analysis and Results

11.1 Introduction

The most important part of any project is the analysis and results derived from it. The Ground Combat Simulator Project being a large and unique project in its size and nature, the level of effort being put in also leads to many open ended results and questions. The areas covered are many, and we will be discussing them one by one.

11.2 Terrain Generation

11.2.1 Analysis

The terrain engine being the most visual part of this project provides an intriguing and engaging view to an open terrain. This terrain is generated with help of first creating height maps and then going through the process of multi texturing. Our terrain has 6000 * 6000 pixels size with total four raw files (each of 3000*3000 pixels). For better performance, resolution and real touch we have set the size of each quad to maximum 20 pixels. However, there are aspects which need more attention and can be even further improved. For example, with some effort the visual aspect of infinite terrain can be added, thus making it more realistic. This can be done using two techniques. One is DEM (digital elevation maps) of area of interest can be gathered from Internet. Second is, the use of algorithms like Perlin Noise and Fractals mathematics to randomly generate terrain maps at run time and can provide the illusion of an Infinite terrain. Weather elements like wind effect can be generated using particle engines. Clouds in this engine are 2D. In order to create infinite terrain, 3D clouds are must.

11.2.2 Results

The performance of this terrain engine is very much to our expectations if this simulation is being executed at the system with required specifications. It really give very natural look and can compete with many of the visual aspects of any commercially available terrain engines. It is because most the terrain engines are of static nature. This terrain is resource friendly even being a major part of project.

11.3 All Purpose Camera

11.3.1 Analysis

Camera plays very important role in any game or in any graphics project. Better the performance of camera more the user of that game /simulation enjoys. We have covered all the aspects of camera to so much detail that its movement of any kind or views of different locations are perfect. However the only point requiring improvement can be the encapsulation of its bare bones functionality in a more programmer friendly class structure.

11.3.2 Results

The Camera being one of the strongest features of this project fulfils almost all of the requirements ever needed for any graphics project. At present the camera code is too rough. With some effort it can be made more user friendly.

11.4 Tank module

11.4.1 Analysis

For ground combat scenario we have **.3ds** models of tanks on both sides i.e. foxland and blueland. As greater number of polygons adversely affect the efficiency in terms of performance. Hence, by making use of the tool Accu Trans Tank is divided into different parts like turret, barrel, gun and body to enhance the performance of simulation.

The aspect of tank movement was successfully managed for rotation of the tank at 360 degrees. As far as tank movement is concerned it is also done so that tank can move freely forward, backward, turn right/left. Tank response to terrain features and the tank firing is also being accomplished. All aspects of firing are not covered due to the shortage of time.

11.4.2 Results

Although by using ACCU TRANS tool we divided tank into different parts like turret, barrel, gun and body etc, but it is creating problems of synchronized rotation and movement of all these part together as well as individually. The aspects of tank rotation/movement were successfully done only for one tank which is part of the troop of tanks at the side of defences. Where as, rest of tanks have automated moves. However, one must understand that if one tank is done properly then other can also be done, but this aspect was intentionally left aside due to the apposite of time.

Tank response to terrain features like the bumpy areas/small ridges is clearly visible, But the other aspects like collision detection, gunner/commander/driver views and target acquisition are not covered. We have just shown that the tank is firing, although all aspects of firing (like aiming, calc range, loading rounds etc) are not covered due to the shortage of time. It is quiet obvious that simulating only a tank is a complete project in itself. If we would have started on research of tank mechanics then I personally think that our simulation might not be having any limits.

11.5 Missile Firing

11.5.1 Analysis

We have used a model of missile which is being fired by the enemy aircrafts. This all needs mathematical modelling to depict that the missile is being fired at target. The firing simulation of missile is calculated mathematically using Bezier curve equation, however further improvements can include the induction of effect of air resistance, wind speed and type/ shape of missile.

11.5.2 Results

Although we achieved bare minimum results, however, major effort at the mathematics end is required and there is very less literature freely available. The present simulation is able to simulate the behaviour of missile with minimum mathematical computation, however a complete research project can be undertaken to simulate the behaviour under all situations. Further improvements in this field can include behaviour of other guided weapons and long distance weapons.

11.6 Air Craft Movement

11.6.1 Analysis

Three enemy aircrafts in arrow head formation. These aircrafts appear on the screen from enemy territory, they move on fixed x and y axis only values for z axis are manipulated during their translation. After firing missiles on targets they fall back on some axis. Although this is not much as far as aircraft’s mechanics and dynamics are concerned.

11.6.2 Results

We have incorporated these aircrafts just to depict the enemy air superiority. It is quiet obvious that it’s very much basic simulation of aircraft. But further improvements can be done by smoothing of the flight paths using the Bezier curves and other related mathematical equations, which will make the flight even more realistic. However, the technique becomes more computational and heavy. There is no end to improvements and this project is a very humble start. A complete research project can be dedicated to simulate the behaviour of a realistic aircraft in bomber and fighter roles.

11.7 Resource Bank

11.7.1 Analysis

This project would have been worth nothing without great and huge collection of valuable resources. Resource collection and creation is most important aspect that needed major chunk of project's time and due attention to the smallest details. Content wise it is the richest area of this project, and most visible as well. If there is any flaw in the visual accuracy of any model/content then it is immediately visible, thus such lapses cannot be compromised upon. We also have created a library of sound samples from FMOD audio API so that various war sounds can be added at appropriate time and locations.

11.7.2 Results

We have created very selective and fine resource bank for this project. We only faced problem in realistic modeling of human (military soldier) models. Even then the resource bank creation for such like project can be compared to any of the contents of a professional project. However, it is said that there is always a room for improvement hence small little enhancements can be made in the areas such as texturing for 3D models, textures for the particle engine (if included), Tank modeling to detailed levels, improvement in aircraft and missile models and above all the induction of human models.

Here it is also worth mentioning that our terrain module is best outfit of the project and most hard earned resource. Sound samples are also unique collection of it’s own.

11.8 Sound Effects

11.8.1 Analysis

We have made use of FMOD to create the sound effects of different elements of war as well as nature. FMOD is cross platform audio engine. One alternative was OpenAL, another cross platform audio API. There are lot more API’s and audio engines available in the market to add sound effects of any kind. Here in our simulation we have chosen the FMOD audio engine as better choice with lot of variety. FMOD is subdivided in two APIs; FSOUND and FMUSIC. We have made use of FSOUND API, which include the samples and streams.

11.8.2 Results

At the background sample sounds are added as shown in table 11.1.

Table 11.1 Sound Samples

a. Helicopter sound samples.
b. Tank movement and firing.
c. Arty gun and mortars firing.
d. Wind/breeze sound samples.
e. Aircraft sound and missile fire.
f. Sound sample of overall battle scene.
g. Small Arms fire effects i.e. Infantrymen rifle/machine gun

Addition of sounds made our work more presentable and attractive. Use of FMOD API code is very easy and programmer friendly. However, the field of audio has very large variety available in the market and one can improve upon by adding number of other sound effects.

11.9 Menu Design

11.9.1 Analysis

Basically we require menu so that our simulation could be user friendly. Menus are computer programs. They guide the user step by step through procedures to be followed. They enable the Tanks and other weapons to firing readiness. We have made use of fonts to built our menu system.

11.9.2 Results

Through these menus we can start the simulation, view particular area of terrain, move through whole terrain i.e. use of camera controls, can manipulate the blueland forces tanks, automate the enemy tanks, launch an air attk and carry out the fire of missiles, tanks and rifles. We can also move through different sub forms.For the reason of simplicity and friendliness we have used ***FONTS*** to implement our menu design. We also have the option to view different results of the battle but these are to very limited scale.

11.10 Development of Battle

As defender we must see for possible enemy actions and plan own counter actions. This development of the battle plan is game of mind sets for the commanders of both sides. Possible en actions and own counter actions are shown in table 11.2.

Table 11.2 Development of battle

Enemy Actions	Own Counter Actions
(1) Enemy commences operation with a brigade strength supported by squadron of tanks by first light D-day to reach main defences by last light D-day. En is expected to use combat aviation and air power for the early clearance of security zone.	**(1)** Employment of covering tps so as not to allow the en to have a free run. Covering tps battle positions would be based on mutually sp def localities(kidney bunds) supported by minefields and will force the enemy to pre selected killing zone.
(2) Enemy will make use of air attks extensively to destroy our mobile elements. Where as Enemy e armour will attempt to avoid running battle with our covering troops and would aim at fixing and by passing it.	**(2)** Any attempt to bypass our forward battle positions will be thwarted by fighting a successive battle, adjusting posns or moving to altn posns. Also request for close support missions against enemy tanks can be used.
(3) On DCB, the enemy armour squadron will search for gaps and will attempt opportunity crossing. Failing to exploit the gaps the armour will get deployed on broad front and will engage the main defences.	**(3)** After the covering troops handover battle to the main defences they will occupy position along the DCB and in depth to cover the gaps and supplement the main defences.
(4) En tks conditions the AA on night D-day/D+1 and would reach main def before first light D+1. En will utilize D+1 for recce and probing attacks.	**(4)** We will use/call upon additional artillery support , combat aviation and fighting patrols to deny enemy close observation of main defences.
(5) On night D+1/D+2 the enemy will launch attack across the main defences on DCB to make a bridge head. Enemy will look for opportunity crossings on canal.	**(5)** As per direction of enemy attack readjust def positions, call for arty sp and launch spoiling attack by fire or raid on likely enemy (FUP) forming up place.
(6) If enemy succeeds in breaching the main defences will try to consolidate his positions thereby reorganising along the ditch thereafter efforts would be made to hastily contact the depth positions.	**(6)** In case of loss of any position, launch a local counter attack. We will continue to provide on spot resistance on the DCB and cause enemy unacceptable attrition through all available means

11.11 Summary

Project covered many aspects however, for improvement sub areas can be taken as projects. If work is continued in this field, lot of efforts and economy can be saved.

Chapter 12

12. Future Work and Conclusion

12.1 Introduction

Military simulations play very important role in the training of soldiers. So there is always a need for the improvement in the development of existing simulators. Here the purpose was to just provide the basic idea to UG students that military simulations can be developed to any limit which one can think of. With same perspective in mind, some of the future work is given in next paragraph.

12.2 Future work Plan

We have tried to provide a basic platform for the further development in the field of military based simulations so that our future training programs can be exercised on these simulators. However there is always a room for improvement and we recommend following future work for improvements in this project

12.2.1 Although lot is being done with regards to tank mechanics however, we would like that tank can be taken as separate module and one odd group can simulate it with all characteristics that our Pak Army tanks are capable of performing in the battle field.

12.2.2 To animate the soldiers in true sense we have to carry out future work on the development of such soldier models which can have true behaviour as the actual soldiers have in the battle field. For this we need to explore the field of character animation.

12.2.3 Terrain generation at the run time to be worked either by integration of GIS systems or getting the terrain images at run time from internet and then transforming them for the use in simulation. Another option is this that we can have variety of terrain formats particular to our country or sub continent and then at run time we may be able to choose the particular type of terrain and the afterwards make the settings for the echelons of battle on both sides.

12.2.4 Collision detection of moving elements is required to be implemented. As far HLA compliance concerned we need a complete interface in between simulation and the RTI. Reason for this is that we tried to work on this aspect but due to the shortage of time and beyond the scope of UG project we could not do so. However, lot of efforts a put in and almost 25% - 30% of the work is completed. Rest of effort can provide us with JNI (Java Native Interface).

12.2.5 Last but not the least is that this simulation must have the events AI base so that results can be calculated, stored and finally displayed.

12.3 Conclusion

The Ground Combat Simulator project covered many aspects however, room for improvement remains in many of the sub areas and these can be taken as full fledged degree projects of their own accord. If work is continued in this field, lot of efforts and economy can be saved. No doubt, all modern armies of the world are using simulations at tactical and strategic levels to train their soldiers as well as commanders at all levels. If any equipment used in ground combat is too costly or fragile to be used excessively in field then it can be simulated. Many a situations cannot be generated in training grounds realistically. At tactical and strategic levels, situations cannot be generated at all, as it requires thousands of troops, and machines. Therefore, need is to choose this new technology as soon as possible, and use it to its full potential. If we are successful in achieving this level of proficiency, only then we stand a chance to compete with the modern armies of the World. As far as HLA implementation is concerned, lot of work and efforts are put in by our group which can be further continued to have a good interface in between simulations of any nature and the RTI.

BIBLIOGRAPHY

[1] How to Become Guru in Shorter Time,
Release 1.0 Author(s) Peter Toft, DMI.

[2] Richard S. Wright Jr. and Michael R. Sweet, *OpenGL SuperBible, Second Edition.* New York: Waite Group Press, 1999

[3] ICIB (Infantry Company In Battle) GSP- 1801

[4] IBIB (Infantry Battalion In Battle) GSP- 1808

[5] Major Operations Of War (Defence) – Pamphlets

[6] www.gametutorials.com

[7] www.3dcafe.com

[8] Dave Shreiner, *OpenGL Reference Manual.* New York: Addison- Wesley Publishing Company, 1999

[9] Jackie Neider, Tom Davis, and Mason Woo, *OpenGL Programming Guide.* New York: Addison-Wesley Publishing Company, 1994

[10] NeHe Tutorials for programming in OpenGL.