Open-Source Code for 3D Visuals

M. Tech - Stage 1 report

Submitted in Partial fulfilment of the requirements

for the degree of

Master of Technology

by

Lt Col Chetan Dewan

(06305403)

Under the guidance of

Prof. Sharat Chandran

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

July 2007

Acknowledgement

I would like to thank Prof Sharat Chandran for his invaluable support, encouragement and guidance. He is a constant source of inspiration and his guidance has made my endeavour an exciting experience.

Lt Col Chetan Dewan

Abstract

While the Increase in the computing and 3D graphics power of the PC over the last decade has been incredible its contribution towards visual simulation to train soldiers has not seen the same trend. Modelling and Simulation have not reached a level in the Defence forces where they become a regular part of military training. Due to increasing costs of training with live ammunition, most modern armies of the world are going in for entity as well as wargaming simulators for training. Attempts are being made at all levels to increase their acceptance as training aids as well as proliferation. In order to help achieve this, most modern armies are trying to create their own customised training simulations. Commercial tools used for creating these simulations are not easy to customise besides having an extremely expensive licensing policy. These drawbacks can be overcome by developing a customised simulation development toolkit that would be easy to use, expandable to future requirements, free from vendor lock-ins. There are two approaches to this problem either to write a toolkit from scratch or customise a suitable general purpose game engine in an open-source environment, allowing the defence forces to use it at no extra cost and easily modify it to meet their requirements as and when required. Creating a toolkit from scratch would be too expensive and time consuming therefore customising an engine would be a better choice. A study of popular open source game engines revealed that no open source engine met all the armed forces basic requirements in its native form, therefore suitable projects would have to be combined and then customised to add additional functionalities to the native engines. This type of customisation would permit future upgrades and cater to changing requirements. The toolkit would comprise of several popular open source projects thus having the "best of open source" and also the strength of the community supporting it allowing the engine to be easily and cheaply improved over time. This report presents our introductory concepts related to training simulators in the armed forces, and talks of our perceived simulation toolkit and the modules it should have. The report further chalks out our plan for creating this customised simulation development toolkit.

CONTENTS

Chapter-1

INTRODUCTION

From movies to games to simulation systems, the power of 3D computer graphics has increased tremendously over the last decade (Fig 1.1). Computer graphics can be used for visual simulations to improve perception of soldier thus imparting effective training. Creating these customised video games requires tremendous programming skills as also an advanced simulation system. In this chapter we take a Macro view of simulations as used by modern day armies, comparison between some advanced video games and simulations, proprietary technologies used, their related problems and what the difference is between war-game simulations and entity simulators. We end this chapter by giving an overview of the topics that are going to be covered as a part of this report.

Figure 1.1: Computer Generated Graphical Images

1.1 Wargaming and Simulation

As the practice of warfare is radically changing and becoming more sophisticated, war gaming and simulation are being increasingly used to predict the outcome of war to carry out tactical planning and low cost training of soldiers. During World War II, the German army regularly war-gamed operations [1] in much the same way as that is done by many modern armies of today. The manual war-games used by the Germans were very similar in style to current games. (The Germans considered their games military secrets and were not made available to civilians). These war-games were used as a command and control tool to predict the outcome of battles even before they were actually fought. This gave the commanders at all levels some foresight

of the battle even though they were highly inaccurate and unreliable. It was soon realized that these war-games, despite being relatively imprecise and ambiguous, had usually been accurate enough to be useful and that they worked. With the development of technology in the late 80's a new era of computer based war-gaming and simulations emerged which were more scientific and somewhat more accurate and better suited to modern day warfare.

Many modern armies' are building and using computer simulations to test equipment designs and tactical scenarios. The most notable among this is the US Army's DARPA (Defence Advanced Research Projects Agency), project to conduct simulations on a computer network and allow the various war-games and entity simulators to interact with each other in a "virtual world." The background for the DARPA project centers on the joining of two distinct types of simulations in distributed networks exercises. The first is a system simulation, or entity simulator, which refers to computer systems which model operator control with graphical displays on a monitor or cockpit simulators which are full-scale models of the system of concern. In this simulator a single vehicle entity is modelled, and its actions are directly controlled by an operator in real time, often by input devices such as joysticks or mock-up controls which are true replicas of the original controls. Sometimes these simulators are mounted on a DOF(degree of Freedom) robot controlled by a dynamics model to give the feel of motion as well. To help perceive this kind of simulator better Fig 1.1.1 shows an advanced prototype of an entity driving simulator with major parts marked.

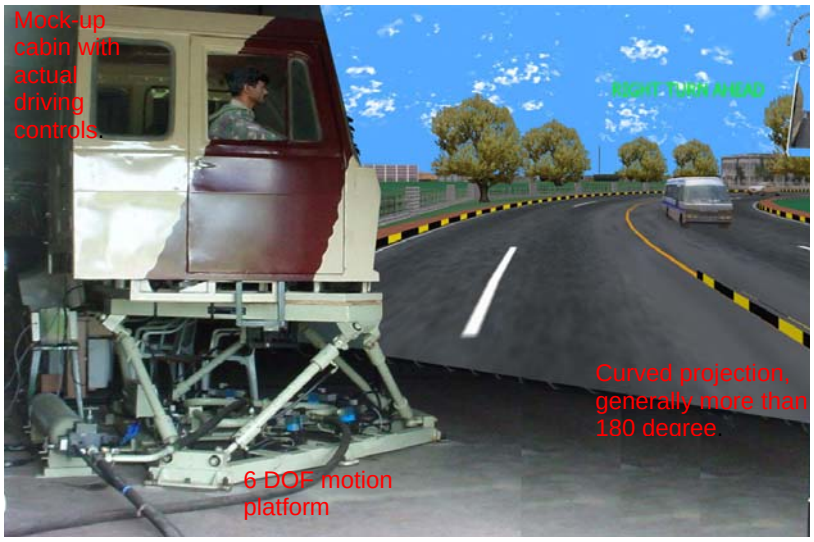

Figure 1.1.1: Prototype driving simulator

The second simulation type is the exercise simulation or war games simulator, referring to computer programs which model the interaction between forces such as army divisions. The forces modelled by these war-games are controlled by operators representing force command staffs. These operators in turn follow orders given by actual commanders at various different levels. Control instructions, which may or may not be in real time, are input by key board commands, and results of interaction between entities are determined by programmed probabilistic relationships. Advanced games are multiplayer where commanders control different groups of forces, one group is designated as own force and the other as enemy force. A well known US war-game is ModSAF. Both types of simulations model a virtual world in which the forces or entities operate. Distributed simulations use a network to bring entities represented by different computers into the same virtual world. Initial distributed simulation efforts focused on entity simulators, but further efforts such as the DARPA project mentioned above attempt to merge the entity and war-game simulators in the same virtual space.

The Combined Arms Tactical Trainer (CATT) and Aviation Combined Arms Tactical Trainer - Aviation Reconfigurable Manned Simulator (AVCATT) facility is one such simulator which is a networked suite of simulators designed to replicate the interiors of armoured vehicles, for example, Challenger main battle tanks, Warrior armoured fighting vehicles and Scimitar light tanks. The facility is used to train a variety of army personnel from the foot soldier, going into battle as an individual, through to the brigade commander, who could be responsible for hundreds of lives. CATT is the largest and most sophisticated virtual training facility in the world and because of this has gained a listing in the, "Guinness Book of Records". The simulators are housed in a building the size of two football pitches in Warminster which, in turn, is able to be linked in real-time to a sister facility in Germany. The CATT system enables crews to view a realistic computer-generated world through armoured vehicle periscopes and then fight a battle against a 'virtual foe', also generated by computer. Commanders plan and view the exercise from Battlegroup Headquarters simulators. To make the exercise as effective as possible the system has been made extremely realistic. For example, engines overheat if left idle for too long, repairs are needed if vehicles are damaged and supplies have to be brought up if the battlegroup is to keep fighting. Mobile Infantry commanders can disembark from an armoured vehicle simulator and then climb into a linked 'infantry' simulator to continue the battle 'on foot', exactly as they would do in a live situation. Fig 1.1.2 shows the CATT complex and some arbitrary views of simulations.

Figure 1.1.2: The Combined Arms Tactical Trainer (CATT) and the (AVCATT)

The 'players' in their simulators can fight against other 'players' in simulators or computer-generated forces. They can exercise within a site or between the UK and German sites. Following the exercise the whole 'battle' can be played back for post-exercise analysis. CATT provides more effective training, as current simulators lack the complexities of a modern battlefield and only involve small parts of a battlegroup rather than a whole formation.

There is also a group of simulators which don't fall under the category of war-games but are a no of entity simulators grouped together to give collective training to a bunch of soldiers at once. These simulators have limited tactical training ability, these combine entity simulators to create scenarios of low intensity conflict like Counter insurgency operations, guerrilla warfare and other hit and run tactics. They are also used for small arms training on a Firing range like environment. Fig 1.1.3 shows collective training on one such simulator.

Fig 1.1.3 Shows troops training for laying an Ambush.

1.2 Commercial games Vs Military Simulation.

The commercial computer game industry has made giant technological leaps in the past few years. These advances have been greatly enjoyed by the general public, yet they have not been used to their full potential [2]. Many of today's computer games have better graphics, sound and interfaces than the most advanced military simulations, and are much less expensive. Fig.1.2.1 shows scenes from popular games Half Life 2 and Quake 4.

Fig.1.2: Snapshots from popular games Half Life 2 and Quake 4

However, most games in the commercial sector fall short when it comes to artificial intelligence (AI). Until very recently, AI was never a part of the game that a developer would spend time on, because it was the graphics and fun factor that sold games, not great AI. People are beginning to demand improved AI, and for good reason. Once a player has determined the behaviour of the computer forces, the game becomes easy to beat, the player can defeat the computer the same way every time, so the game ceases to provide a challenge.

The other drawback being that the models used in games do not behave very realistically. A vehicle driving game would not have the required vehicle dynamics incorporated for that particular vehicle model like effect of sudden braking, acceleration etc. If the game is a first person shooter then bullet would generally travel in a straight line like a laser beam and not follow the trajectory as an actual bullet would, also the effect of environmental factors on its

trajectory are ignored which have considerable impact on the designated point of hit of any projectile. Generally speaking in the case of a aiming kind of game with say a model rifle provided to shoot down stationary targets, the point of aim would be the same as point of hit of the bullet and increasing target distance would not have any real impact.

Seeing the huge advances in the gaming industry it was at first thought that it would be beneficial to the forces if we could harness the power of the commercial games' outstanding graphics and intuitive interfaces, and integrate an advanced AI and model behaviour into such a game, so that it could be used for training. Therefore a study of many kinds of games from first person shooter like Quake3 to strategy games like Microsoft's Close Combat war game where the user commands a group of forces, leading them against enemy forces was carried out. Also a feasibility study was carried out in an attempt to determine the possibility of incorporating an advanced AI and model behaviour into one or more of them. However it was soon realised that even though some of these great games had editing tools, but none gave us the flexibility we would need to improve the AI or model behaviour. The only way to change the AI and model behaviour would be to change the source code, which we did not have access to freely. Therefore the studies concluded that the best solution would be to customise a freeware game engine and create a state of the art gaming engine suitable for making military simulations. It was decided that some of the great ideas incorporated in commercial games could be borrowed and customised to suit military needs. Adding a non rule-based, intelligently interactive AI engine and advanced model behaviour would allow personnel to train against a computer adversary that behaves like a human adversary or drive a vehicle entity simulator that behaves like the actual vehicle on a particular terrain.

1.3 Problem statement

The objective of the project is to decide the essential and desirable (wish list) features that we would want as part of a 3D rendering engine suitable for military simulation and then carry out a survey of the many freeware game engines available on the Internet. Based on our survey we would then try to customize this engine to our requirements' and finally test our project with a sample simulation.

1.4 *Previous Work*

As part of this ongoing project an open source rendering engine Coin3D was selected and an importer for the *OpenFlight* 3D visualization format was created. This importer was a converter utility to convert OpenFlight files to the native Coin 3D format, Open Inventor (IV). Due to enhancements of many open source projects over time it is now seen that a number of them can either directly use *OpenFlight* format or have built in importers and exporters to this format. Direct use is beneficial to us as then we will not be required to develop a modeling tool for this format. If the database conversion is done there is no suitable open source 3D model editor, available for the Open Inventor visual database. Further the Coin 3D project is not very popular with the gaming community and hence has stagnated over time, seeing very little development. Therefore it was decided to look for an alternate popular project which would be easier to customize and enhance.

1.5 Overview

The overview of the report is as follows. In chapter 2 we will list out our design philosophy and then list our essential, desirable features and finally look at perceived broad technical issues to our approach. In Chapter 3 we will try and craft out a desirable broad architecture for this customized 3D rendering engine. In Chapter 4 we list out some suitable rendering engines based on our survey and finally select the engine, which gives us the maximum benefit. Chapter 5 describes the future roadmap for this project.

Chapter-2

DESIGN PHILOSOPHY

Commercial off the shelf simulation systems are priced heavily and applications built on them require runtime licenses costing up to six figures for each application built. However, in reality, a review of some of these simulations reveals that almost all of them have essentially the same features; probably 90% of the functionality each provides is basically the same as that of a game, with minor differences. The difference between all of them essentially boils down to the most advanced features each provides. However, these advanced features are not needed for the vast majority of simulations. Worse, once these systems are used, projects become locked into the proprietary technologies used as the code is written expressly for the proprietary engine. In case you want a modified application due to changing requirements, you have to go back to the vendor for an upgrade as no source code is available.

There is another problem using these proprietary systems: Even if source code is available for the application there is no way to modify the underlying Engine if it doesn't meet the current needs. Developers can request a feature from the vendor, but such requests are rarely answered and almost never in a timely enough manner to be useful to the current project. This requires significant developer time and effort to build "workarounds" to overcome problems with proprietary tools. As mentioned before, most of these systems had close to 90% in common with games being sold commercially at a handful of the cost. Despite this, commercial system builders are still charging as though they provide a unique product, only available from them.

In order to overcome these shortcomings of commercial systems for building simulations and games, we have come up with a multi part philosophical credo [4] on which the 3D rendering engine will be based.

- Keep everything open to avoid vendor lock-ins and increase flexibility.
- Use a modular design so that anything can be swapped out as technologies mature at different rates and anything can be added.
- Make it as generic as possible since it is not known what application it's going to have to support next.
- Use the power of the open source community so that end users are not burdened with future development and enhancements to meet future requirements.
- Support reusability of existing code and models already made.

These simple principles will be used as our guidelines when making decisions throughout the selection and enhancement process as described below.

2.1 Essential features

Keeping our multi part credo in mind a list of essential features are listed in the order of priority that we would like in the native open source 3D rendering engine. These features would aid us in selecting a suitable engine for modification.

- Rendering Engine: The 3D Engine will be an abstraction over Open GL and or Direct X for the Graphical component. Operating System specific components should be centralized and separated so as to make the engine code platform independent as far as possible. The rendering engine should have a solution using Visual Studio with .net support. It will also have facilities for rendering 2d content, such as text and skins for the interface, to the screen using any freeware tools. It should be upgradeable in the near future.

- File format support: Support for popular file formats including exporters and importers for the *OpenFlight* scene description format. *OpenFlight* is MultiGen-Paradigm's [5] native 3D content is the leading visual database standard in the world and has become the defacto standard format in the visual simulation industry. Since the defence forces have a no of previously made visual databases in this format it will permit reusability of existing models.

- Audio Engine: Sound and music in games and simulations is extremely important to give a realistic feeling. The open source rendering engine will use the Open Audio Library (Open AL), [9] which is a software interface to the audio hardware. It resembles the OpenGL API in coding style and conventions and uses a syntax resembling that of OpenGL where applicable. The interface consists of a number of functions which allow a programmer to specify the objects and operations in producing high-quality audio output, specifically multichannel output of 3D arrangements of sound sources around a listener. Consequently, legacy audio concepts such as panning and left/right channels are not directly supported. OpenAL does include extensions compatible with the IA-SIG 3D Level 1 and Level 2 rendering guidelines to handle sound-source directivity and distance-related attenuation and Doppler effects, as well as environmental effects such as reflection, obstruction, transmission, reverberation etc.

- Physics Engine: Physical laws govern the objects in the real world. A suitable Physics engine is required in order to realistically model several physical phenomena, such as joints, springs, damping devices (e.g., shock absorbers), friction, gears, motors, and collisions. These are important for simulating vehicles, objects and human models in virtual reality environments. A physics engine also handles the effects of gravity, velocity changes, inertia correctly, as well as collision [8] with other objects placed in the world.

- Character Animation: To animate characters the engine should have support for a skeletal based 3-D character animation library [10] . It should have exporters and importer plug-ins for most popular 3-D modeling packages. It should allow characters to transition smoothly between different animations, such as walking and running to get a wide variety of movement characteristics.

- Scripting support: The engine should have some form of scripting support as a scripting language allows changes in game behaviour with minimum of programming on the developers' part.

- Hardware device support: The engine should have support for basic keyboard and

mouse control of objects with feature to add third party libraries for control by VR haptic devices and motion trackers.

2.2 Desirable features

There are a host of desirable features that come to mind which when incorporated will make the engine more versatile, easier to use as also extend its capabilities. Some of the features are listed below.

- Graphical Scene editor: This will greatly aid in placing objects at desired locations in the rendered scene and manipulating them as and when desired. The manipulations can be done visually.

- Terrain and vegetation rendering methods: These algorithms can help in generation of a synthetic terrain by picking up objects such as trees, shrubs etc and placing them randomly in the scene helping in automatically generating a geotypical terrain.

- Advanced Hardware optimisation: The engine should be able to use latest advances in hardware. This can be implemented by using libraries like Intel performance primitives (IPP), and other specific libraries from NVIDIA and ATI etc. These libraries contain performance-optimized functions for digital media, data processing applications which offers thousands of optimized functions covering frequently-used fundamental algorithms in:
    - Video coding
    - Signal processing
    - Audio coding
    - Image processing
    - Speech coding
    - JPEG coding
    - Speech recognition

  These libraries help get the most out of today's Multi-Core Processors and advanced graphic processing units (GPU's).

- Advanced environmental features: The engine can have advanced features like varying lighting conditions by changing time of day, moving clouds, effect of wind etc.

- Particle system editor: A editing system for generating special particle effects such as rain, fog, dust, smoke or importing such effects from other games and using them will greatly add to the realism of the simulations.

- Record and playback capability: Recording and playback capability is an important feature of almost all military simulations. This gives very important after action review that aids in correcting mistakes made by users of the simulation.

- 3-D model designer and viewer: This would form an important part of any system, which would aid in expansion of the assets library by adding new models.

- Networking support: There should be readymade socket support for TCP/IP, UDP etc. It should be able to support distributed interactive simulation (DIS) [12]. A run time infrastructure (RTI) can be created for this purpose.

- Advanced hardware support: Inbuilt support to read data/cues from built in computer hardware such as serial ports, parallel ports, USB, fire wire, game port with

configuration options via a graphical interface. Support for generic motion trackers.

- Advanced motion models: The engine should have readymade support for motion models such as ufo, drive, spin, game, walk, etc.

- Satellite and GIS data support: The engine should be able to render a terrain based on elevation data/GIS data available in standard formats. It should be able to drape satellite imagery on this elevation map to create a geo specific terrain.

- Advanced Intersectors: Ready made collision detection library to include Intersectors that can be attached to objects to detect or maintain constant height above terrain, bump mapping, checking line of sight, ground clamping, xyzpr , zpr etc.

- 3D Model Polygon optimizer: There should be a system for optimizing the no of polygons in any 3D model being used as part of the simulation such that its basic shape and characteristics remain unaltered resulting in an optimal frame rate.

- Support for Level of details(LOD): As the object is depicted farther and farther away it can be made up of reduced no of polygons to show its basic shape and reduced size. As the object is brought nearer its size can go up and no of polygons can be slowly brought back to normal.

- Advanced Audio support: There should be inbuilt support for Stereo and 3d sound with very advanced effects like environmental audio and Doppler effects.

- Advanced lighting effects: It should support various light lobes and light points, environmental lighting, background illumination etc.

- Road tool: There should be a tool for generating a random network of roads of various kinds from highways to by lanes on the click of a button with auto texturing. Developer should be able to add other features such as streetlights, footpaths, speed breakers etc from a customizable library.

- Night vision: There should be capability to convert the terrain into a night scene at the click of a button to be used with special IR projectors suitable to be viewed through night sights of weapons.

2.3 Implementation Strategy

After going over commercial projects on the net as well as open source projects it was realized that the rendering engine would be an impossible task for a single developer as a MTP project. This kind of project is generally done by a well-construed project team and takes several years to complete and can go into many millions of lines of coding. The only way to successfully do this kind of project would be to use the projects from the open source community and do only coding required to combine the power of these projects into a single solution. This would require a very thorough survey of open source rendering engines and other projects that we would use as add-ons to our solution set.

API of the base-rendering engine, which we would select, should be easy to use and easy to enhance/modify as per our changing requirements. Since our project is primarily for MS Windows the strategy to enhance the base-rendering engine is best based on a DLL (Dynamic link library) model. Open source projects selected, as enhancements to our rendering engine should be compilable to DLLs. The users of the rendering engine should not have to concern themselves with handling of low-level details. They should be able to make the objects do

whatever they want it to, and the engine should handle any low-level interactions (positioning, rendering, checking collisions, etc.) that occur. However, if the users so desire, they can always modify the API if it doesn't handle something the way it is desired to be handled. The engine should be able to import/export to a number of file formats so that content creators can use most of their favourite tools, (both commercial as well as open source) to create the 3D content for use in games/simulations. With the above as our basic guidelines we began a survey of the various rendering engines available on the net and made our selection.

Chapter-3

OUR APPROACH

WE decided to start our quest for open source projects from Source Forge and other such Internet sites as a starting place to look for basic rendering engines and add-on modules to add to our open source engine. Our selection would be based on our credo and the following two criteria:

- Project's technical background and merits.
- User support base of the project.

The rationale for choosing projects upon their technical background and merits would help in effective integration with minimum problems while a project's user support base and popularity would ensure that these projects don't stagnate (ie remain current) too soon as they have many "indirect" developers.

3.1 Base Engine Schematic

A rough schematic of our vision of this engine in terms of modules is shown in Fig 3.1.

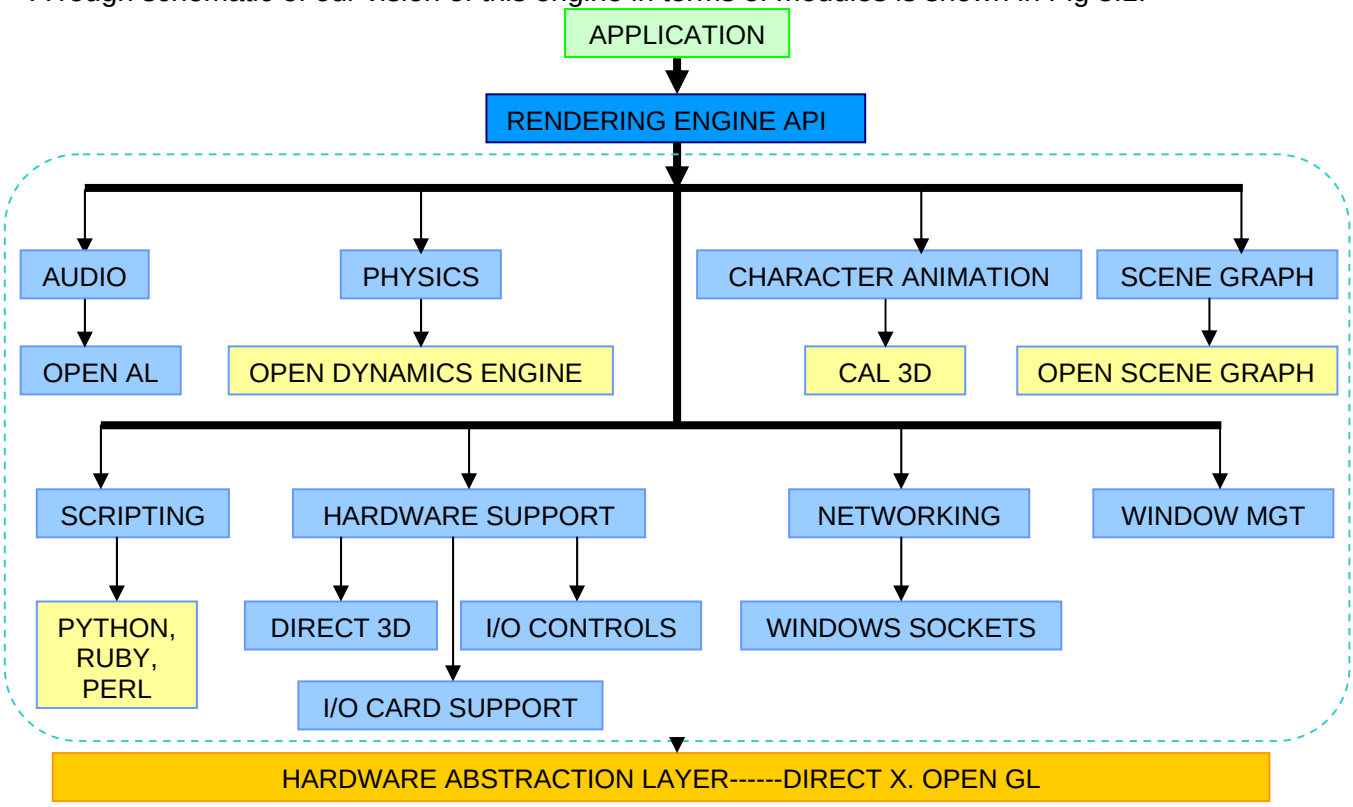

Fig.3.1.1: Rough Schematic of our perceived basic rendering engine

3.2 Desirable Modules

The schematic of fig 3.1.1 contains a broad classification of modules and does not contain modules from our wish list that we would like include in our rendering engine. In addition to the features mentioned above we would also like to add some additional advanced high level features which would make this rendering engine easy to use by the designers for creating artificial objects and make them interact with other objects and display results. For example, it is possible to declare an object that is transformable has physical properties (such as appearance, mass, size, bounding box, animations, etc.) and can be "linked" to other objects. This will automatically cater for low-level details, which will be automatically handled at the engine level. Some of the advanced features that we would like incorporated are shown as a continuation of our schematic shown by Fig 3.1.2

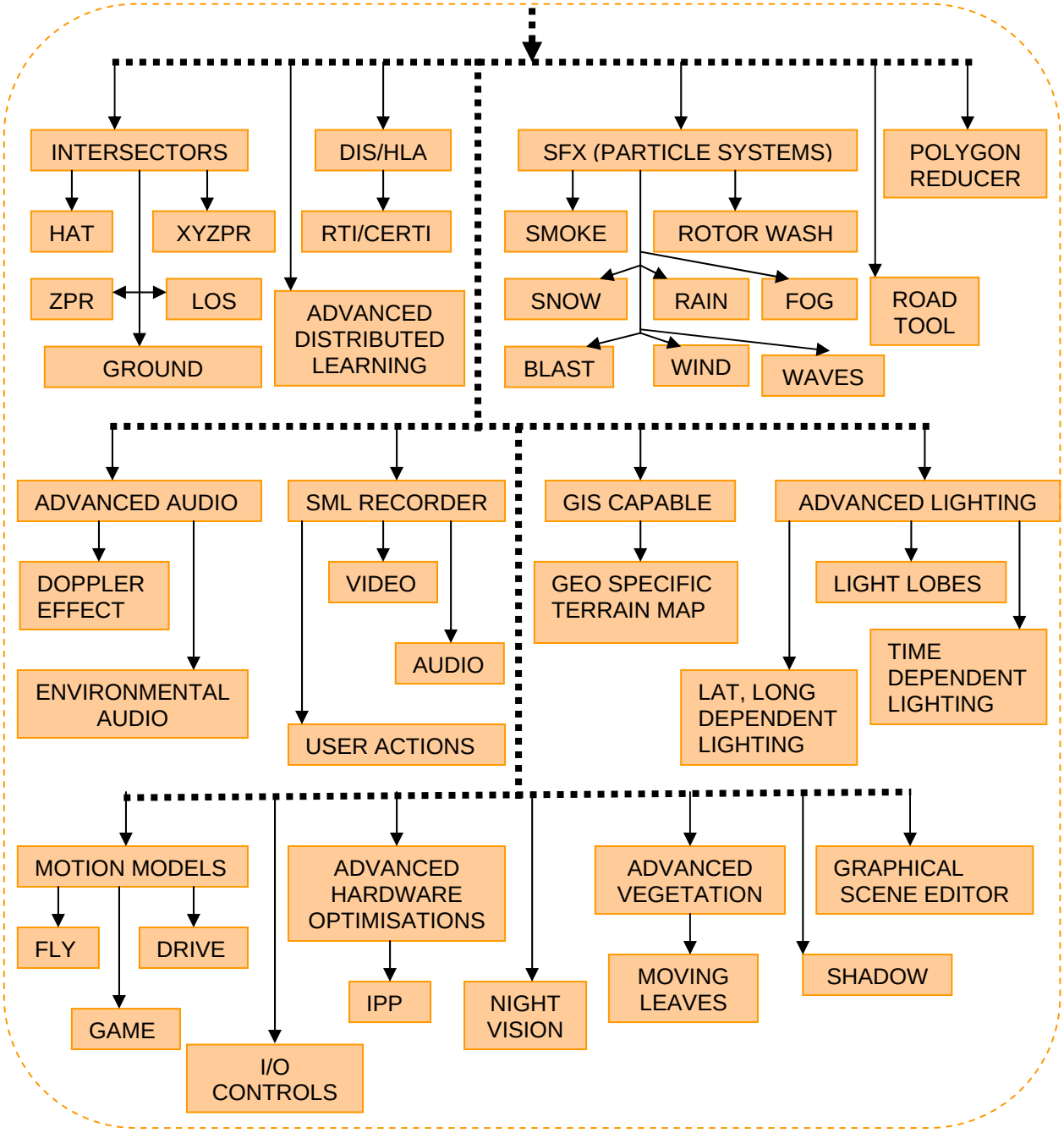

Fig.3.1.2: Desirable Modules of our perceived rendering engine

After finding a suitable base engine we would look for projects that satisfy one or more of the desirable features and finally integrate all the projects into our solution. Once completed our engine would comprise of the base modules as well as most of the desirable modules. Those desirable modules for which we do not find suitable projects will be marked for future development. These can be done as part of the ongoing project.

Chapter-4

OUR SOLUTION

When we began our study our net search showed a large no of listings. It would be a Herculean task to check out all of them, so we embarked on a novel plan. We first segregated the most popular freeware rendering engines that gamers use and also looked for any rendering engines that may be in use for creating any kind of military simulations. In order to further narrow down our search we started looking for rendering engines that had Importers and Exporters to specifically the *OpenFlight* format as the developers had several readymade databases in this visual database format. Not having support for the OpenFlight format would mean wastage of several years of hard work. In the end based on our credo and essential list of features that we had decided on previously we zeroed in on two different rendering engines that made the grade. These are the Open Scène-graph and the Delta 3D.

4.1 Open Scene-graph

The OpenSceneGraph [7] is an OpenSource, cross platform graphics toolkit for the development of high performance graphics applications such as flight simulators, games, virtual reality and scientific visualization. Based around the concept of a SceneGraph, it provides an object oriented framework on top of OpenGL freeing the developer from implementing and optimising low-level graphics calls, and provides many additional utilities for rapid development of graphics applications. This rendering engine was found to be quite popular with gamers and was under active development as was evident from its home page, it could directly use the *OpenFlight* format and had importers and exporters to other popular formats besides it met the tenets of our multi part credo mentioned previously.

4.1.1 Features

- **Productivity:** The core scene graph encapsulates the majority of OpenGL functionality including the latest extensions and a whole set of add on libraries which make it possible to develop high performance graphics applications very rapidly. The application developer is freed to concentrate on content and how that content is controlled rather than low level coding. Combining lessons learned from established scene graphs like Performer and Open Inventor, with modern software engineering methods like Design Patterns, along with a great deal of feedback early on in the development cycle, it has been possible to design a library that is clean and extensible. This has made it easy for users to adopt to the OpenSceneGraph and to integrate it with their own applications.

  For reading and writing databases the database library (osgDB) adds support for a wide variety of database formats via an extensible dynamic plugin mechanism, the distribution now includes 45 separate plugins for loading various 3D databases and image formats. 3D database loaders include OpenFlight (.flt), TerraPage (.txp) including multi-threaded paging support, LightWave (.lwo), Alias Wavefront (.obj), Carbon

Graphics GEO (.geo), 3D Studio MAX (.3ds), Peformer (.pfb), Quake Character Models (.md2). Direct X (.x), and Inventor Ascii 2.0 (.iv)/ VRML 1.0 (.wrl), Designer Workshop (.dw) and AC3D (.ac) and the native .osg ASCII format. Image loaders include .rgb, .gif, .jpg, .png, .tiff, .pic, .bmp, .dds (include compressed mip mapped imagery), .tga and quicktime (under OSX). A full range of high quality, anti-aliased fonts can also be loaded via the freetype plugin.

The scene graph also has a set of Node Kits which are separate libraries that can be compiled in with your applications or loaded in at runtime, which add support for particle systems (osgParticle), high quality anti-aliased text (osgText), special effects framework (osgFX), OpenGL shader language support (osgGL2), large scale geospatial terrain database generation (osgTerrain), and navigational light points (osgSim).

- **Portability:** The core scene graph has been designed to have minimal dependency on any specific platform, requiring little more than Standard C++ and OpenGL. This has allowed the scene graph to be rapidly ported to a wide range of platforms - originally developed on IRIX, then ported to Linux, then to Windows, then FreeBSD, Mac OSX, Solaris, HP-UX and even PlayStation2!

- **Scalability:** The scene graph will not only run on portables all the way up to Onyx Infinite Reality Monsters, but also supports the multiple graphics subsystems found on machines like a mulitpipe Onyx. This is possible because the core scene graphs supports multiple graphics contexts for both OpenGL Display Lists and texture objects, and have been designed to cache rendering data locally and use the scene graph almost entirely as a read-only operation. This allows multiple cull-draw pairs to run on multiple CPU's, which are bound to multiple graphics subsystems. Support for multiple graphic context and multi-threading is all available out of the box via osgProducer - all the examples in the distribution can run multi-pipe just by use of a simple configuration file.

4.2 Delta 3D

Delta3D [3], [4] is an open source game and simulation engine Specifically designed for military applications and interacting with tools the US military commonly uses. It has been built atop other open source projects licensed under the Lesser Gnu Public License (LGPL). Considering that this engine was made with the primary aim of making military simulations, this engine meets almost all our basic requirements and maybe some of the advanced features we are looking for.

4.2.1 Features

Most of the features are very similar to OpenSceneGraph as it uses a number of libraries that OpenSceneGraph uses. Delta3D has been used as the underlying architecture for several games and military as well as civilian simulations [11], most of them in the serious games arena. Delta3D also provides some basic facilities to support the development of AI using its high level scripting language. With all its advanced features it is not for novice programmers and requires team-based development. The API schematic of this engine is shown in Fig: 4.2.1. Delta 3D is an active and popular Open source project that is under rapid development. Even though the website boasts of a plethora of features, use of some of these features shows that they are crude and still require reasonable development effort.

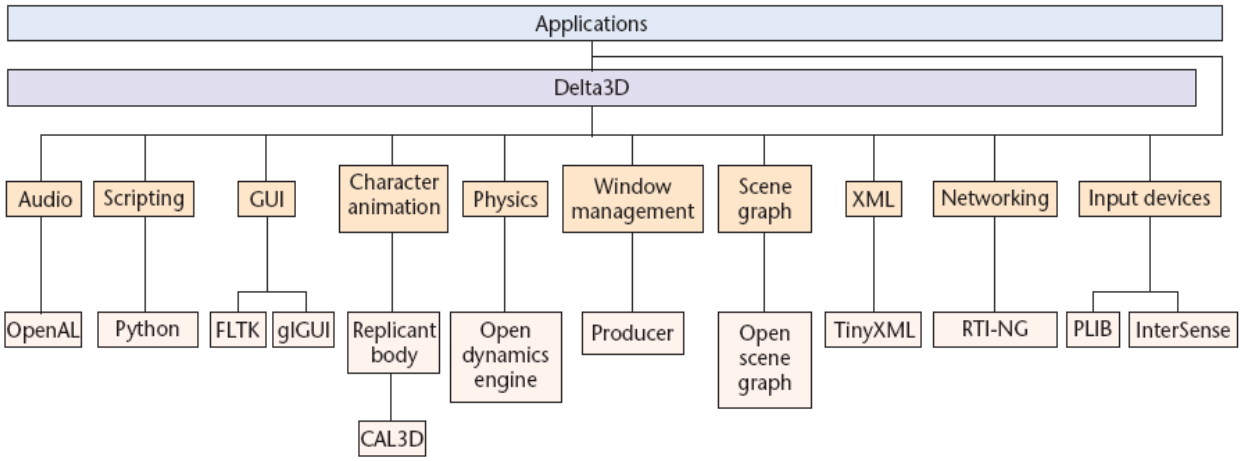

Fig.4.2.1: A block schematic architecture of Delta 3D

4.3 Our choice

Further study of both the rendering engines revealed that the Delta 3D was originally designed for military simulations and may be better suited to meet our requirements. From the Block Schematic available from its website it appears to meet all our basic requirements and some of the desirable features as well. It is also possible to add new projects to the delta 3D Visual studio.net solution to enhance features and capabilities desired. It would require writing of a very few lines of code to customise it to meet many of the desirable features we mentioned in previous chapters.

Chapter-5

FUTURE ROADMAP

The objective of the project is to build a 3D Rendering Engine which will be used for developing and deploying Entity simulators as well as making war gaming simulations in the future. It will also provide advanced distributed learning (ADL) capability for training and monitoring the level of expertise of the soldiers in the future. The final goal is to provide an application development toolkit which is simple to use, expandable in the future and open source. The engine so developed should be expandable to the same class as world class rendering engines such as Lynx Prime from Multigen Paradigm [5]. The engine will use a number of popular open source projects so that it can be continuously enhanced with the help from the Open Source community and always remain current. It should be easy to add and remove modules to give additional capabilities to this engine.

5.1 Review of Work

The present project will be completed in three phases:

- **Phase 1:** The following will be done as part of this phase :-
    - Creating a basic design architecture based on our philosophical credo.
    - Enhancing this design with the desirable features thus creating an overall enhanced architecture.
    - Finding a open source rendering engine that meets requirements in the same priority as listed in our essential features.
    - Studying this rendering engine in detail to find out suitable ways of enhancement.
    - Testing this engine for stability, direct use of open-flight format and other popular supported formats.

- **Phase 2:** The following will be done as part of this phase :-
    - In this phase we will compile this rendering engine to produce both a debug as well as a release version using Visual studio.NET.
    - We will also search for open source projects that can add desirable features to our base engine.
    - After adding these open source projects to the base engine we will again try to compile the code and debug it where required.
    - We will also segregate those enhanced features for which suitable open source projects are not available and list them for future development.
    - On successful compilation of the modified engine source test simulations using human Models, Lighting, Collision detection, motion models will be used for comparison with our Benchmark rendering engine Lynx Prime to see, what

further customizations and changes in design need to be done to achieve similar quality.

- An attempt will be made to work out an algorithm for random population of a terrain with objects like trees, rocks etc and other such objects which can be placed randomly on any terrain. A pseudocode for the same will be written. If time permits a module to carry out placement will be implemented with a sample database. This kind of module will help in generating geotypical terrains quickly and save on modeling effort.

- **Phase 3:** The following will be done as part of this phase :-

  - We will in this phase try to create a complete simulation of a very basic simulator through which a number of features of this engine will get tested.

5.2 Work Plan

Fig 5.2.1 below shows the plan for phase -1. We design a basic architecture for our rendering engine based on our credo and as per priority of features listed we start studying various open source rendering engines available on the net and create a comparison chart to see which meet max of our desirable feature listing. Check out their documentation if any to seed which would be easier to understand and modify.

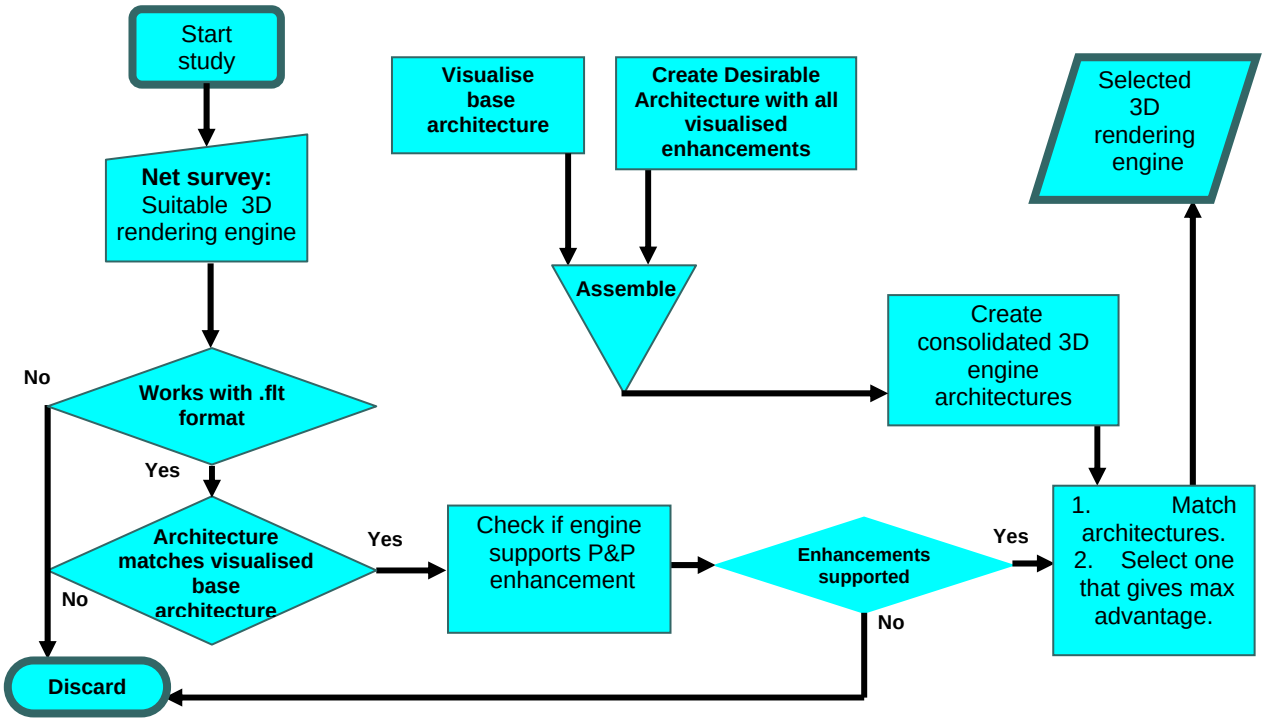

Fig.5.2.1: Plan for Phase -1.

We then try to add-on to this architecture with a list of desirable features and do a simultaneous comparison with our selection chart to see if any of our desirable features are present by any chance. Our final decision to select the rendering engine would be based on both inputs of essential (mandatory) and desirable features. This Phase requires approx 2 months for finalisation.

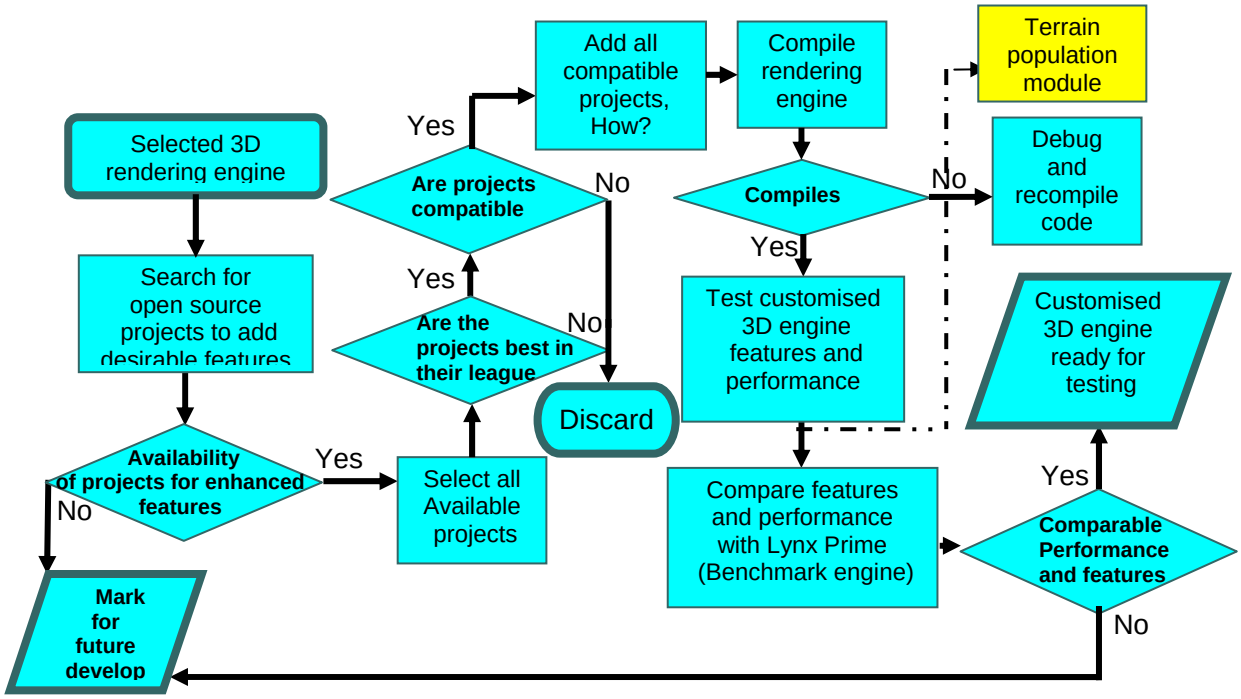

Fig.5.2.2: Plan for Phase -2.

Fig 5.2.2 above shows the plan for phase -2. In this Phase we add additional projects to our Visual Studio solution such that most of the desirable features get incorporated. We also segregate those features which require elaborate development or development from scratch and mark them as separate projects for future enhancement and out of the scope of the present project. We also compare our project with other similar projects. We then work out an algorithm for random population of a terrain with objects like trees, rocks etc and other such objects which can be placed randomly on any terrain. A pseudo code for the same will be written. If time permits a module to carry out placement will be implemented with a sample database. This kind of module will help in generating geotypical terrains quickly and save on modeling effort. This phase will take approx nine months.

Fig 5.2.3 below shows the plan for phase -3. It is essentially completion of Phase –2 and testing of project with sample simulations. On completion of we will attempt to create a complete basic simulation. A prototype for demonstration will be created initially and refined by adding some more advanced features. Finally essential documentation would be created for this engine. This phase is expected to be complete at least one week before the proposed date of completion (PDC) for the project.

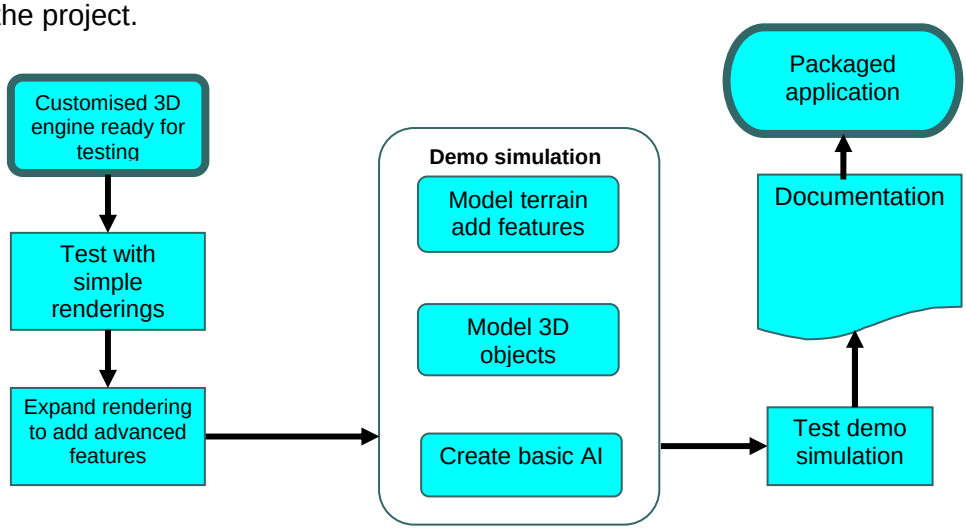

Fig.5.2.3: Plan for Phase -3.

5.3 Conclusion

In this report, we have explored some fundamentals of War-gaming and Simulation and the concepts of entity simulators. We have also explained the difference between a commercial game and a military simulation and made a broad schematic for our customized rendering engine. Next, we reviewed some freeware engines available from the net and made our choice based on our philosophical credo and features we desire. We also checked out several sites with open source projects that have been used to enhance AI and other functionality in games. Finally we chalked out a plan for achieving our customized solution.

REFERENCES

[1] James F Dunnigan. The Complete Wargames Handbook, Chapter 1, Retrieved June 2007.
URL http://www.hyw.com/Books/WargamesHandbook/Contents.htm

[2] Steven M. Alexander, David O. Ross and William J. Lloyd. Developing Intelligently Interactive Computerized Wargames. Pentagon reports, Final report Feb 2000-01

[3] Delta 3D features, Retrieved Mar 2007.
URL http://www.delta3d.org/article.php?story=20051209133127695&topic=docs

[4] Perry McDowell, Rudolph Darken, Joe Sullivan, Erik Johnson; Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems.

[5] Multigen Corp. OpenFlight Visual database format, Retrieved June 2007.
URL http://www.multigen.com/products/standards/openflight/index.shtml

[6] Soren Hannibal. 3d engines for games, a broader perspective, Retrieved June 2007.
URL http://www.gamasutra.com/features/20001013/hannibal_pfv.htm.

[7] OpenSceneGraph Home Page, Retrieved
May 17, 2007 from URL http://www.openscenegraph.com/index.php

[8] Open Dynamics Engine Home Page, Retrieved May 20, 2007 URL http://ode.org/

[9] Open Audio Library Specifications Page,
Retrieved May 20, 2007 URL http://www.openal.org/oalspecs-specs/x44.html.

[10] Character Animation Library 3D FAQ Page, Retrieved May 20, 2007
URL http://download.gna.org/cal3d/documentation/api/html/

[11] Lawrence Rosenblum and Michael Macedonia Projects in VR. The Delta3D Open Source Game Engine, Volume 25, Issue 3, May-June 2005 Page(s): 10 – 12, Retrieved May 2007.

[12] James Hardt and Kevin White. Distributed Interactive Simulation (DIS)
EEL 4781 Computer Networks, University of Central Florida, Fall 1998, Retrieved July 11 2007
URL http://www-ece.engr.ucf.edu/~jza/classes/4781/DIS/project.html

Links to Web Sources

1. Comprehensive graphics toolkit URL
   http://www.quantum3d.com/products/software/archive-do%20not%20post/vtree.html
2. Intel performance primitives (IPP); Retrieved May, 2007
   URL http://www.intel.com/cd/software/products/asmo-na/eng/302910.htm
3. Delta 3D ; Retrieved May, 2007
   URL http://www.movesinstitute.org
4. Wargames ; Retrieved June, 2007
   URL http://www.strategypage.com/military_photos/200522323.aspx.
5. Defence Advanced Research Projects Agency ; Retrieved June, 2007
   URL http://en.wikipedia.org/wiki/DARPA.
6. Aviation Reconfigurable Manned Simulator (AVCATT); Retrieved June, 2007
   URL http://www.peostri.army.mil/PRODUCTS/AVCATT/.
7. Game; Retrieved May, 2007
   URL http://en.wikipedia.org/wiki/Half-Life_2
8. game ; Retrieved May, 2007
   URL http://www.quake4game.com/
9. Freeware game engines
   URL http://www.thefreecountry.com/sourcecode/games.shtml
10. Few hundred game engines
    URL http://cg.cs.tu-berlin.de/~ki/engines.html