Simulator Oerlikon Anti Aircraft Gun GDF-005

By

Capt Khawar (Leader)

Capt Subhani

Capt Imran

PC Umar Khokhar

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
May 2005

Abstract

Training is a very important aspect of any organization, and a huge amount is spent on realistic training of troops. However the highly technical equipments like Antiaircraft Guns, Aircrafts, of tanks have a very heavy running expense even in the peace time, and can cost heavily in terms of resources. Also the training Equipment is not always available, and till now Pakistan Army, Navy or Air force does not have any true Working Simulator. The simulators being used are not very realistic in terms of visual accuracy, level of detail and options available in terms of tactical scenarios.

To over come all these problems, and as a starting point, one specific instance of an Antiaircraft Weapon, being one of the most costly equipments available in Pakistan Army will be simulated with certain limitations.

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 would like to acknowledge our advisor Lec Arif Raza (NUST) for his guidance and for keeping his faith in our abilities. We would also like to thank our esteemed instructors who helped make us who we are and nurtured our capabilities, giving us the skills and the confidence to take on mountains and emerge victorious.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AAA – Anti Aircraft Artillery

AAD – Army Air Defense

AADC - Army Air Defense Command

BMP – Bit Map

DLL - Dynamic-link library

FPS - Frames per second

FU – Firing Unit

FCU – Fire Control Unit

FOV – Field of View

GUI – Graphical User Interface

JPEG - Joint Photographic/Picture Expert Group

LCD - Liquid Crystal Display

PPI - Planned Position Indicator

Chapter 1

Introduction

1.1 Aim

The aim of undertaking this project derives its roots from the need to train the troops of Army Air Defense. The aim is to provide necessary tools for the training of a specific anti-aircraft system based on visual tracking and for the training of personnel who will use the system. The project objectives include creation of a general purpose system in the frame work of the specific instance of one of the most expensive weapons of Pakistan Army. Reasons for selection of Oerlikon GDF- 005 are that it is one of the most expensive weapons in terms of Cost / running expanses (Per piece cost more than 170 million pak rupees).It is very fragile equipment, requires extensive maintenance even in peace time, hence a simulator is required to replace the actual training operations. Only a single simulator has been exported by the government on a huge expanse, which can guide us in creating another simulator for the weapon with certain limitations, as well as expected improvements in other areas, saving cost of export of another. This will be a guide line to develop further simulators of other advanced weapons like tanks / missiles etc.

1.2 Detailed Objectives

It was intended to simulate the basic functionality of anti-aircraft gun Oerlikon in local mode, along with Simulating target tracking and engagement practice of aerial and targets. Other Objectives include the creation of a General purpose terrain engine in which weapon system can be places at will. Creation of an all purpose Camera, which can

become the base for an all purpose simulator, simulating the functionality of any weapon of Pakistan Army. Implementation of Oerlikon specific behavior, including Firing behavior, Rates of movement, Menu System, True to scale models of weapon system, and True representation of Gun Sight.

1.3 The Necessity of an Environment Generator

The essentiality of the simulator may be gauged from the fact that the simulator must be tested against a number of demanding parameters including frame rate, target acquisition time and acquisition accuracy.

1.4 Long Term Objectives and Benefits

Long term Objectives include Realistic training of troops, Creating a basic modifiable simulator for long term, next generation weapons, in process of induction to the organization, Creating a soft ware base on top of which hard ware can be added on availability of funds, Acquiring a functional product at the fraction of the cost of original product.

Chapter 2

Terrain Engine

2.1 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.

2.2 Parts of a Terrain Engine

A terrain engine generally consists of parts, like terrain generation, trees, Foliage, Sun, Moon, stars, Water Bodies and Clouds

2.2.1 Terrain Generation

The terrain generation has been carried 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 if figure 2.1. To tile a second texture on top of the first one to

give the appearance of more detail. This is called detail texturing, which can add a great deal of realism to a scene. Multitexturing is used to achieve this neat effect

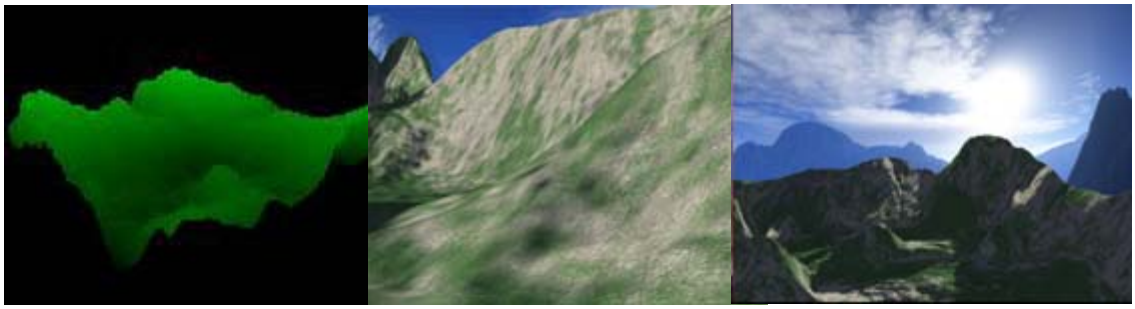

Figure-2.1 Snap Shots of Virtual Terrain

First 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_STIP 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 Multitexturing 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.

2.2.2 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. So how does an alpha value affect what gets drawn in a window on the screen? 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. Alpha values can also be used in the alpha test to accept or reject a fragment based on its alpha value. 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. Thus one can use alpha blending to create a translucent fragment that lets some of the previously stored color value "show through. 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. If one look at an automobile, for instance, its interior has one piece of glass between it and your viewpoint; some objects behind the automobile are visible through two pieces of glass.

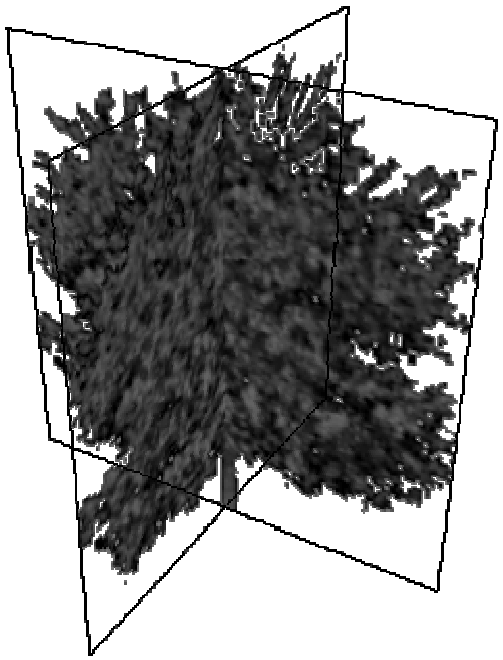

Figure-2.2 2D Tree Image Cross-section View

2.2.3 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 2.2. The Tree is nothing but two quads over which am alpha blended or masked image has been pasted. The result is however neat. With this low polygon solutions 1000 trees were drawn at a reasonable frame rate. These trees cover an area of 2000 m only. 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.

2.2.4 Sun, Moon and Stars

Location of Sun and Moon is calculated based on time of day. Moon date is calculated using a simple formula giving an error of 1 /2 days. Stars are generated randomly, and a total of 200 stars are scattered at night. Sky Color Changes with time of day. Blue at day and grey at night. Available sun light changes with the time of day, reduces at night.

2.2.5 Water Bodies

These are created with tillable textures. A total of Only 9 textures being used for animation, giving a smooth shimmering water effect. A minimum water level is set, below which terrain also changes color. Rivers can be generated easily by “forging” deep gullies in the terrain.

2.2.5 Clouds

These move in 2 layers circling the area, moving at a very slow pace, providing difficulty of vision to the gunner while tracking the aircraft. A very important feature in real time situations, as the Aircraft merges with the cloud cover. For optimization 2D cloud generation carried out. Distance and angle of presentation give illusion of 3d clouds.

Chapter 3

All Purpose Camera

3.1 Introduction

It's impossible for people to move around physically in a "virtual 3D world", therefore there was a need to use a "camera" to get orientated. With the help of camera 3D scene can be navigated and can be looked around in all the directions. When setting up a basic camera in OpenGL there are two main subjects to be taken into consideration, Setting up the view volume and Initializing the camera/view.
In actual there is no camera in OpenGL. There is only one library (glu) that provides a function i.e. gluLookUp () which implements the functionality of camera. But the problem with this function is that its variable cant directly be manipulated to get the 6 degree of freedom. Since the requirement was to build a camera which could implement the behaviour of any of the ground or aerial object a new camera was created which could meat all specific 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 as shown in figure 3.1. One can say that the position point and the target point form a view-vector. The "up" or "tilt" decides if the camera is tilting.

The view volume of camera can be set up in OpenGL by establishing the view angle in y-axis view angle in x-axis far clipping plane and near clipping plane, as shown in figure 3.2. This functionality is also missing in OpenGl. It is also provided by the glu library.

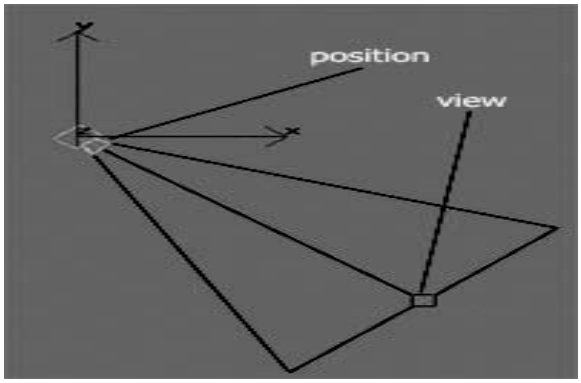

Figure-3.1 Basic Camera Positions

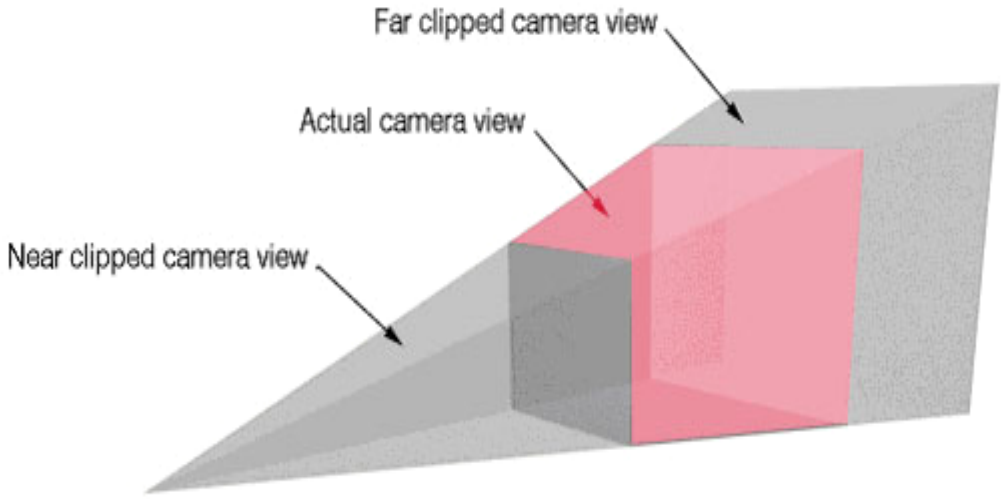

Figure-3.2 Camera View Volumes

3.2 Camera 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. One is able to set each of these matrices to different modes to achieve different effects, for

example, 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. These matrices are used for camera control. In order to change the settings, one have to tell OpenGL that one want 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 for instance, glTranslate.

The gluLookAt function call is worth investigating: In application program these 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 ca 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 3.1.

3.2.1 Detail of Camera Control Command

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. 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. Subsequent transformation commands affect the specified matrix. Note that only one matrix can be modified at a time. By default, the modelview matrix

Table 3.1 Camera Control Commands

Command	Description
glLoadIdentity	Initializes 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
glPopMatrix	Restores the previous camera.
glGetFloatv	Used to Query the OpenGl about the current matrix values
gluPerspective	Sets the view volume of camera

is the one that's modifiable, and all the matrices contain the identity matrix. The command gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz) defines a viewing matrix and multiplies it with the current matrix. The desired viewpoint is

specified by eyex, eyey, and eyez. The centerx, centery, and centerz arguments specify any point along the desired line of sight, but typically they're some point in the center of the scene being looked at. The upx, upy, and upz arguments indicate which direction is up ie tilt.

The glRotatef function rotates the scene by a specified amount in terms of angle in degree about the specified axis. glTranslatef function translate the scene by a specified amount along the specified axis. 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. 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. glGetFloatv(GLenum pname, GLfloat * params) returns values for simple state variable in OpenGL. It is used to query OpenGl about the value of current matrix. The pname parameter is a symbolic constant indicating 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. 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 3.3; its value must be in the range [0.0,180.0]. aspect is the aspect 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. They should always be positive

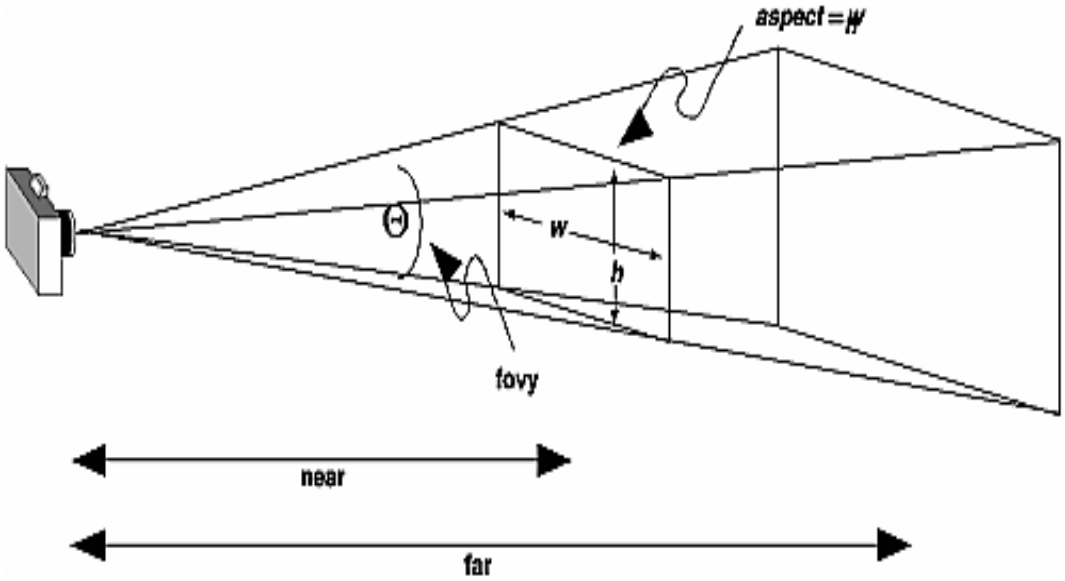

Figure-3.3 Camera Field of View

3.3 Rendering Transformations

Since the models are stored in model space some transformations was required to show them on screen. Basically there are three sets of geometric transformations which are Modelling transforms, Viewing transforms, Projection transforms. Basic model coordinates are shown as in Figure-3.4**.**

3.3.1 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, rotate objects and translate the objects.

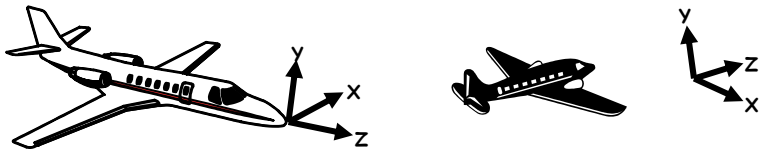

Figure-3.4 Basic Plane Coordinates

3.3.2 Viewing Transform

In viewing transform models are not dealt with. 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

3.3.3 Projection Transform

In Projection transform perspective projections transforms for 3D scene and ortho transformations for 2D scenes is carried out. A *Pinhole camera* model is used for this purpose as shown in Figure 3.5.

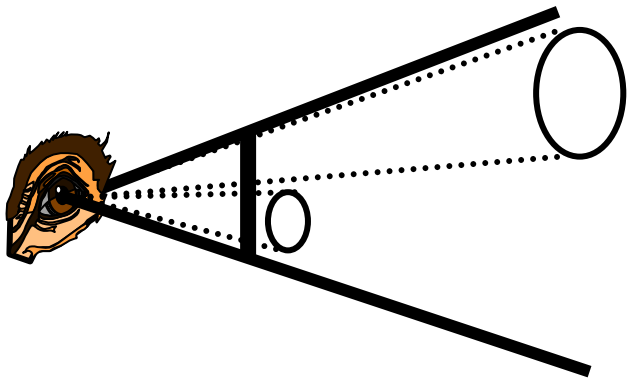

Figure-3.5 Eyes Projections View

3.4 Camera Terminologies

Some of the terminologies related to camera are that a local coordinate system , see Figure 3.8, 3.9, which changes with each transformation and a grand/global, See Figure 3.6,3.7, or a reference coordinate system which remain constant throughout the transformations.

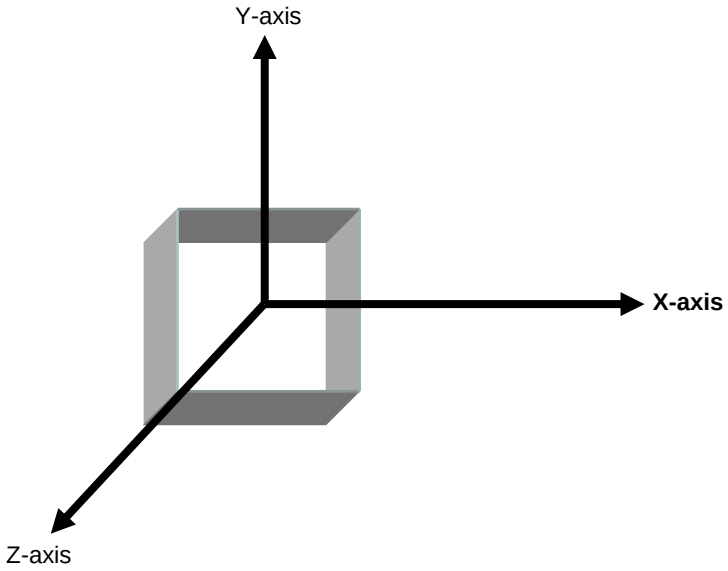

Figure-3.6 Basic Global Axis

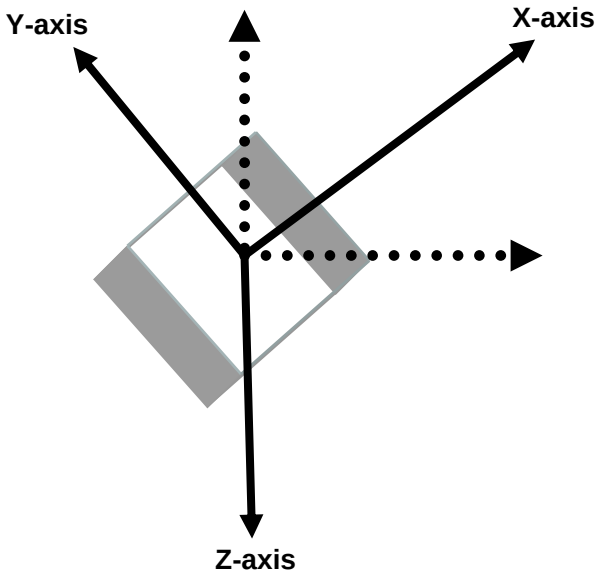

Figure-3.7 Global Axis Rotation

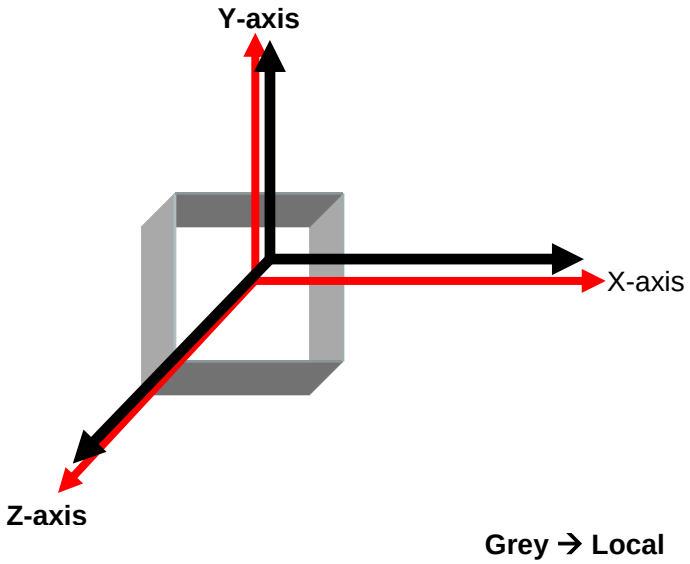

Figure-3.8 Local Axis

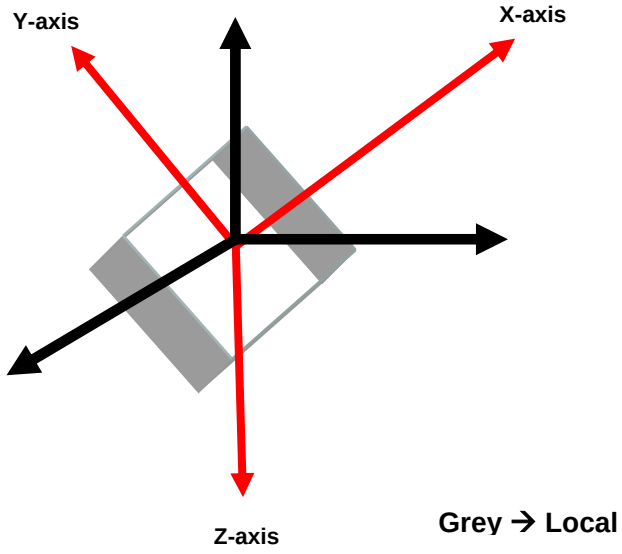

Figure-3.9 Local Axis Rotations

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.

3.5Camera Making Steps

The steps are Creating a transformation matrix, Selecting Matrix Mode i.e. GL_PROJECTION using glMatrixMode(), Setting the view volume using gluPerspective(), Selecting Matrix Mode i.e. GL_MODELVIEW using glMatrixMode(), Saving the state of matrix on top of modelview matrix , stack using glPushMatrix(), Loading transformation matrix in OpenGl as modelview matrix using glLoadMatrixf(), Carrying out required transformation using glRotatef() glTranslatef(), Queering the OpenGl about the current modelview matrix values and loading them into transformation matrix using glGetFloatv(), Restoring the state of saved modelview matrix using glPopMatrix(), Updating the Scene, Rendering the scene.

3.6 Camera Working Methodology

To understand the functioning of camera, understanding of the basic concept and functioning of matrices is important. Matrices being the base of camera are the first step.

3.6.1 Matrices in OpenGL

OpenGl uses column instead of rows to represent vectors in a matrix

$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

In OpenGL, all vertices are transformed by two matrices. These are the modelview matrix and the projection matrix. The projection transformation precedes the modelview transformation. The projection matrix is "responsible" for defining a view volume and clipping to this volume. The projection matrix is setup once and then left. In a normal OpenGL application the modelview matrix may change several times per frame.

3.6.2 Matrices in General

The problem with matrices is that they are used in many applications for many purposes (including 3D graphics). So the idea is to think about using matrices as a tool in different ways when dealing with different problems. When dealing with OpenGL programming it is much simpler to understand rotations/translations if one think a matrix is a coordinate system representation. The OpenGL stack is a stack of matrices; the topmost matrix i.e. the current matrix represents the current coordinate system. A matrix multiplication such as M = M * T can change the current coordinate system from where One is positioned. “Translation matrix” and "rotation matrix" definition will make it more clear.

3.6.3 The Modelview Matrix

In OpenGL the modelview matrix is responsible for camera (view) transformations and objects (modelling) transformations, hence the name modelview. One way of thinking about matrices is to think of them as defining a coordinate system. The "properties" of a coordinate system are the axes and the origin. So a coordinate system can be defined by the 3 axes and the origin. At least one coordinate system is required to be used as a reference, think of this system as the "grand"

coordinate system. It is sometimes called "world" space or the World Coordinate System (WCS). It is defined as a "grand" system with the origin at (0, 0, 0), the x-axis pointing at (1, 0, 0), the y-axis pointing at (0, 1, 0) and the z-axis pointing in the direction (0, 0, -1). Now the matrix can be used to define this coordinate system with the 3 axes in a 3x3 matrix:

1 0 0 <- the X axis

0 1 0 <- the Y axis

0 0 1 <- the Z axis

To define the origin the 3x3 matrix is enlarge into 3x4 matrix:

1 0 0 <- the X axis

0 1 0 <- the Y axis

0 0 1 <- the Z axis

0 0 0 <- the origin

This matrix defines a coordinate system. OpenGL uses 4x4 matrices. These 4x4 matrices allow not only rotations and translation but also scaling and shearing.

The 4x4 matrix is used to build a basic 6DOF moving system. The OpenGL matrix used to represent the grand coordinate system is:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

As long as one uses only translations/rotations the last column will always be 0 00 1. This matrix is called the identity matrix. This matrix has the property that vertices that are transformed by this matrix remain unchanged. So the identity matrix is loaded into OpenGL's modelview matrix stack and then draw the vertices like this:

glLoadIdentity();

```
glBegin(GL_POINTS);
        glVertex3f(1, 1, 1);
glEnd();
```

The coordinates of the drawn point in world space will be (1, 1, 1). If the current modelview matrix isn't an identity matrix all following glVertex (and glNormal) calls will define a position in the current coordinate system, which is defined by the modelview matrix.

Imagine a coordinate system with origin (100, 0, 0), the X axis pointing at (1, 0, 0) the Y axis pointing at (0, 0, 1) and the Z axis pointing at (0, 1, 0). The matrix for this coordinate system is:

1 0 0 0
0 0 1 0
0 1 0 0
100 0 0 1

Now if call is made to the function

```
glBegin (GL_POINTS);
        glVertex3f (1, 1, 1);
glEnd ();
```

Then the point will have the coordinates (1, 1, 1), in the local space specified by the above matrix, transformation of this point results into another matrix which is another coordinate system, having origin at different location.

The matrices can represent coordinate systems to apply to build a 3D movement system. The basic idea behind it is: don't move objects, move coordinate systems. It's done via matrix multiplication. This means matrix is multiplied by another matrix. In case of translation this other matrix is called "translation matrix" or **T** and in case of rotation the matrix is called "rotation matrix" or **R**.

3.6.4 Translation matrix T

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Take an identity matrix and assign the X, Y and Z translation to the 12th, 13th and 14th elements of the matrix, respectively. Let's construct a translation matrix for the vector :

(10, 12, -4)

The matrix looks like this:

1 0 0 0

0 1 0 0

0 0 1 0

10 12 -4 1

So if matrix M represents a certain coordinate system (already loaded onto the OpenGL stack) and say:

M = M * T

where **T** is the matrix above, the coordinate system will be moved 10 units along the **local** x-axis, 12 units along the **local** y-axis and -4 units along the **local** z-axis.

3.5.5 Rotation matrix R

Like translation matrix, Rotation matrix is also used to produce another coordinate system, whose origin is individuals position.

glRotate*$(a, 1, 0, 0)$: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

glRotate*$(a, 0, 1, 0)$: $\begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

glRotate*$(a, 0, 0, 1)$: $\begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

When the current matrix is multiplied with the rotation matrix , a new coordinate system is generated like in translation.

3.6.6 Simple Movement System

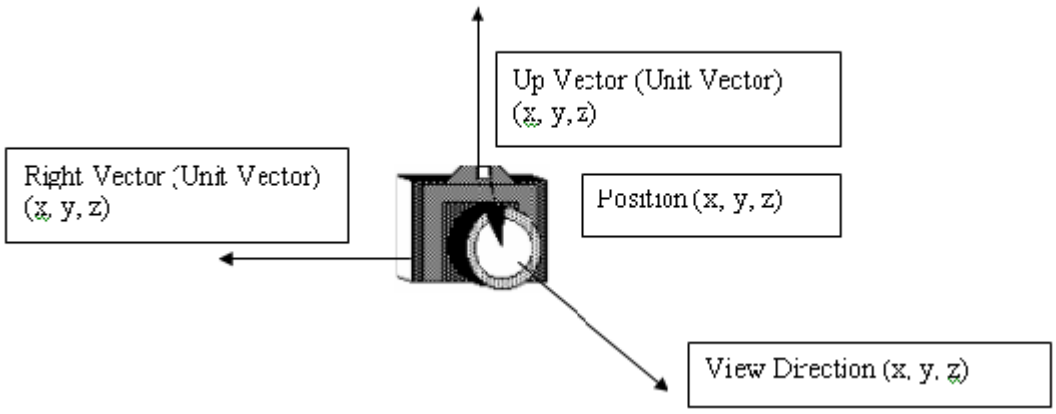

Figure-3.9 Camera's Parameters

The matrices can be used to represent the coordinate system. Now to build a simple 3D 6DOF movement system, the essential features are moving and rotating. Thus the basic interface looks like thus the basic interface looks like Fig 3.9:

```
struct SF3dVector
{
GLfloat x,y,z;
};SF3dVector F3dVector ( GLfloat x, GLfloat y, GLfloat z );
class CCamera {
public:
SF3dVector ViewDir;
SF3dVector RightVector;
SF3dVector UpVector;
SF3dVector Position;
SF3dVector ViewPoint;
float Transform[16];
CCamera(float x, float y, float z);
~CCamera();
void Render();
void rotateGlob(float deg, float x, float y, float z);
void SetViewByMouse(bool sightOn, float elev, float Bg);
void update();
void MoveZGlobal(float distance);
void MoveYGlobal (float distance);
void MoveXGlobal (float distance);
void MoveZLocal(float distance);
void MoveYLocal (float distance);
void MoveXLocal (float distance);
void RotateXLocal (int elev,bool sightOn,float deg , bool GroundMode);
        void RotateYLocal(float deg);
        void RotateZLocal (float deg);
        void rotateGlobAboutView(float deg,float x,float y,float z);
        void RotateYGlobal(float deg);
        void RotateZGlobal (float deg);
```

```
        void rotateGlobal(float deg);

        void Zoom(float ZoomFactor);
};
```

The orientation of the camera is described by three vectors: The *view direction*, the *right vector* and the *up vector*. Initial position of the camera is described 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 vectior (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:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

3.6.6.1 Movement Along All Axes

Let's say, MoveZ(float distance) is called. This movement is related to translation along local z-axis by an amount distance.Movements related to translation like along x-axis , y-axis and z-axis have been dealt with using two different approaches:

```
glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
        glLoadMatrixf(Transform);
```

```
glTranslatef(0,0,distance);
glGetFloatv(GL_MODELVIEW_MATRIX, Transform);
glPopMatrix();
```

In this 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 modelview matrix of current matrix, so that all subsequent transformations affect the modelview matrix .Than the current state of modelview matrix have been saved so that the effect of subsequent operations is not cumulative. Then transformation matrix is loaded on current stack of modelview 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 ModelView matrix).Now after translation have occurred , the values of transformation matrix have been changed , representing another coordinate system.

The transformation matrix in programme is updated by querying the opengl. Opengl returns the current values of the modelview matrix which are stored in transformation matrix. Previously stored modelview matrix is restored then.After all this , camera is updated in each frame using update() function:

```
void CCamera::update() {
RightVector.x = Transform[0];
RightVector.y = Transform[1];
RightVector.z = Transform[2];
UpVector.x = Transform[4];
UpVector.y = Transform[5];
UpVector.z = Transform[6];
ViewDir.x = Transform[8];
ViewDir.y = Transform[9];
```

```
ViewDir.z = Transform[10];
Position.x = Transform[12];
Position.y = Transform[13];
Position.z = Transform[14];
}
```

After the camera is updated the translated scene is rendered using the Render() 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 using translation operation , since only the last row of transformation matrix is affected , the last row is simply added into the specified direction vector multiplied by the amount of distance to be travelled on that particular axis e.g. in this case if move is required along view vector that is represented in programme by the row Transformation[8],Transformation[9],Transformation[10] of transformation matrix.

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 re3quired 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. This function looks like:

```
CCamera::moveGlob(float x, float y, float z, float distance) {
        Transform[12] += x * distance;
        Transform[13] += y * distance;
        Transform[14] += z * distance;
}
```

3.6.6.2 Rotation About All Axis

Now if the function RotateZLocal (float deg) is called , the view is rotated about Local z-axis.In other words the view have been rolled.This function looks like:

```
void CCamera::RotateZ(float deg) {
        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
                glLoadMatrixf(Transform);
                glRotatef(deg, 0,0,1);
                glGetFloatv(GL_MODELVIEW_MATRIX, Transform);
        glPopMatrix();
}
```

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 modelview matrix as current matrix, so that all subsequent transformations affect the modelview matrix. Than the current state of modelview matrix have been saved so that the effect of subsequent operations is not cumulative. Then transformation matrix is loaded on current stack of modelview 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 ModelView matrix).Now after translation have occurred, the values of transformation matrix have been changed, representing another coordinate system.

Now transformation matrix in programme is updated by querying the OpenGl. OpenGl returns the current values of the modelview matrix which are then saved in transformation matrix and transformation matrix is updated. And previously stored modelview matrix is restored. After all this , camera is updated in each frame using update() and then the scene is rendered using Render() function.

Now if the function RotateZGlobal(float deg) is invoked , the view is rotated about Global z-axis. This type of motion is called roll. This function looks like:

```
void CCamera::rotateGlob(float deg, float x, float y, float z) {
float dx=x*Transform[0] + y*Transform[1] + z*Transform[2];
float dy=x*Transform[4] + y*Transform[5] + z*Transform[6];
float dz=x*Transform[8] + y*Transform[9] + z*Transform[10];
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadMatrixf(Transform);
glRotatef(deg, dx,dy,dz);
glGetFloatv(GL_MODELVIEW_MATRIX, Transform);
glPopMatrix();
}
```

This function first uses the formula that first stores the values in variables dx, dy, dz of all the y-values of right, view, up vectors if the rotation is to be carried about Global y-axis. After this is done, then the steps to obtain local rotation become similar to that of global rotation.

3.6.6.3 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. This function is as follow:

```
void CCamera::SetViewByMouse(bool sightOn, float elev, float Bg)
{
        POINT mousePos;
                int middleX = SCREEN_WIDTH  >> 1;
                int middleY = SCREEN_HEIGHT >> 1;
        float angleY = 0.0f;
        float angleZ = 0.0f;
        static float currentRotX = 0.0f;
        GetCursorPos(&mousePos);
        if( (mousePos.x == middleX) && (mousePos.y == middleY) )
         return;
        SetCursorPos(middleX, middleY);
        angleY = (float)( (middleX - mousePos.x) ) / 25.0f;
        angleZ = (float)( (middleY - mousePos.y) ) / 50.0f;
        RotateX(elev,false,-angleZ,false);
        rotateGlob(-angleY, 0,1,0);
}
```

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.

This camera has a unique feature that is missing even from professional cameras ie zoom. Once the Zoom (float Zoom Factor) is

called the scene is zoomed by an amount Zoom Factor. This effect has been achieved by manipulating the first argument of gluPerspective() function of OpenGl , which is used to establish the view volume in OpenGl.

3.7 Camera Features

This camera has many features other than listed in this document. The main features which have been used/implemented in project. Besides this, it can distort the shapes, can scale and also can carry out the sheering effect. The local mode features 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 This camera can magnify the scene to the desired amount. This effect has been achieved by manipulating the first argument ie field of view of gluPerspective();

3.8 Camera Uses

Some example uses of this camera are Simulation of Behaviour of Human Being like an Infantry Soldier or moving Infantry column, Simulation of Behaviour of Ground moving objects like vehicle/tanks etc, Simulation of Behaviour of an Aerial Object like an aircraft etc.

3.9 Gun Movement Simulation

360 degree traverse in both clockwise and Anticlockwise direction and Limiting movement of gun up till 90 degree in elevation and -5 degree in depression. All movements required to implement the gun menu system.

3.10 Aircraft Behaviour Simulation

The camera can Roll, Yaw, Pitch, Flying in all directions with adjustable aircraft speed.

3.11 Simulation of Behaviour of Vehicle/Tank

This camera can simulate all types of possible ground movements. Adjustable speed.

3.12 Simulation of Behaviour of Human Being

In this mode camera can lookup till 90 degree, Can lookdown till -45 degree, Can walk, run and look around.

3.13 Implementation of Collision Detection Algorithm

Collision detection with the ground (terrain) has been implemented using the camera's position and the height returned by the height function of the terrain. At each frame it is tested, whether this camera's position is below or above the terrain. If it is up, then it is fine else camera is moved on to the terrain, all the camera variables are updated and then the scene is rendered.

3.14 GUIde

GUIde is an OpenGL graphic user interface that tries to copy the functionality of the big GUI systems in the world. It doesn't contain pieces of original work and one shouldn't expect it to have amazing features and use stunning technologies. This is more like an experiment in which the author has tried to make this library useful to create small compatible programs and to make GUI for OpenGL games.

This software is probably full of bugs, design errors and naive source code, all of these done with the help of C++, OpenGL and sometimes, with third party libraries.

3.14.1 Design

The Guide design was inspired from Borland's VCL (Visual Component Library). Methods names, events and properties are, in general, the same as the Borland's ones, but the body of functions are written by the author. Where he has used external sources of inspiration, he has tried to mention this thing. It should not be expected that the GUIde controls react in the same manner as their Borland's version, but, in general, this case is the common one. The author has not tried to make a wrapper for the controls; he only kept the minimum quantity of things he thought he needed to make the GUIde functional.

3.14.2 Features

The list of GUIde features isn't impressive at this moment. GUIde is OOP organized, has bugs, isn't documented at all (though author has started to fix this problem), has a small list of controls, supports transparencies, is hardware accelerated and it is a sinkable thing. Because GUIde uses a double buffered system to render its content, a semitransparent front dialog rendered over a scene will be observed in which monsters kills themselves and they can be viewed.. In short, GUIde renders all the visible controls at every frame. Simple clipping systems hide the unnecessary content from user and minimize the rendering effort the GUIde is doing.

3.15 Integration of LCD

LCD is a display which can show the data out put from computer. To display the menu system, which is the complete technical detail of the gun, LCD was required to be integrated. For this purpose a LCD was procured and programmed to perform this job. First of all it was interfaced with printer port using inpout32.dll file. The data was out put to it via printer port at an address *0x378* and using inpout32.dll's function Out32(DATA, string).The LCD was turned on by sending appropriate control words .Then its display was cleared with another code word so that the current display is not affected by the remnants of any previous display .After this its functions were set to the values to achieve 1- 8 bit data output, 2xline display, font = 5x7 dots. The LCD was then prograta matically integrated in programmer's eternal loop to show the appropriate output.

3.16 Integration of Joy Stick

To simulate the behaviour of actual firing yoke, an idea was perceived that if the joy stick is inverted and used to control the gun, its behaviour is exactly the same as that of the firing yoke of the gun. Since this was a simple idea to implement the actual behaviour of the firing yoke of the gun so it was implemented by integrating the joy stick in the project. The winmm.lib has been implemented for the joy stick. After successful Integration of joy stick, its implementation to simulate the behaviour of gun was the next mile stone. This was also achieved using the Camera class's Rotation and Translation functions .Then the sensitivity of the joy stick was controlled so that it exactly simulate the firing yoke of the gun. By sensitivity control it means that once the joy stick is moved a little, gun movement should be a bit slow and if the joy stick is moved to fully traverse the gun, the gun movement should be

sudden and fast .All these affects are achieved and the joy stick is successfully integrated in the project to simulate the behaviour of the actual gun firing yoke.

3.17 Gun Movements in Menu System

In the implementation of the gun technical details like gun behaviour at start up , alignment , checking of fire sector and laser sector, setting of gun position just before loading and to test the gun drive control system and the sight control system automatically , all these checks required and involved gun movements .For this purpose a general purpose function i.e. MoveGunTo() was made which has been programmed to move the gun in traverse and elevation. This function is programmed in such a way that the gun always follows the shortest path to reach a particular point. Different points are stored at run time in different menu systems like fire sector and laser sector to mark the sector .Then MoveGunTo () function is called with the array of stored points, and this function automatically moves the gun to the exact location of the stored points, in the order they were stored following the shortest path to the next stored location, to carry out the check whether the fire sector e.g. has been marked correctly of not.

Chapter 4

Gun Firing Simulation

4.1 Introduction

The basics of the motion of a projectile in 2D space can be best described as If a projectile is fired vertically upward into the air with an initial velocity of $v0$ m/sec from a point $s0$ meters above the ground, then (neglecting air resistance) after t sec the projectile is $s = s(t) = -1/2\ gt2 + v0t + s0$ meters above the ground(see Figure 4.1), where $g = 9.7$ m/sec2 is acceleration due to gravity. If the projectile is fired at an angle of elevation of θ with respect to the horizontal at an initial velocity of $v0$ m/sec from a point $y0$ meters above the ground (see the figure to the right), then (neglecting air resistance) after t sec the projectile is located at the point whose coordinates are given parametrically by $x(t) = (v0 \cos \theta)t$, $y(t) = -1 / 2\ gt2 + (v0 \sin \theta)t + y0$.

In other words,

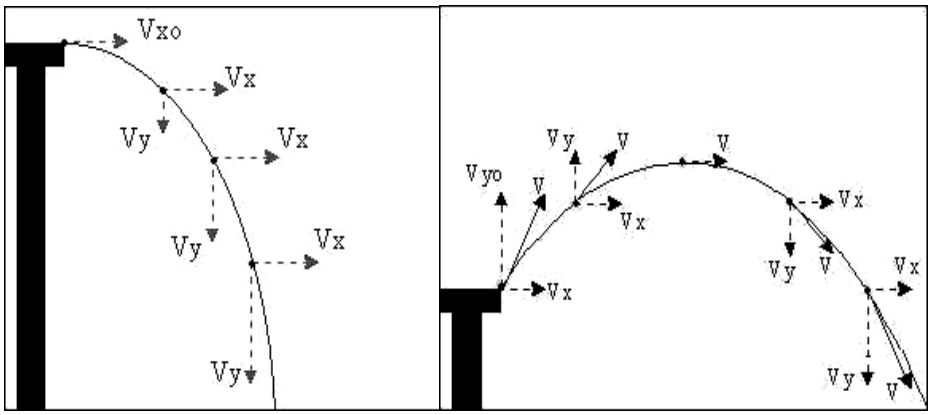

Figure-4.1 Projectile Motion Vectors

Once the object leaves the table, it experiences a downward acceleration equal to gravity. Thus the vertical velocity (Vy) is continually increasing. The horizontal velocity (Vx) remains constant and is equal to Vxo. The two vectors Vx and Vy are added together to get the velocity at each point on the path. If an object is pointed at an angle, the motion is essentially the same except that there is now an initial vertical velocity (Vyo). Because of the downward acceleration of gravity, Vy continually decreases until it reaches its highest point, at which it begins to fall downward.

4.2 Computer Representation

The same formulae can be best describes in computer algorithms as follows:

a is the angle of projection

u is the initial velocity

g is the force of gravity

t is the time after initial projection

y is the height of the projectile

x is the horizontal range of the projectile from origin

Giving:

$$y = ((u * \sin(a))*t) - (0.5 * g *(t^2))$$

$$x = u * t * \cos(a)$$

4.3 Difficulties.

The main problem was that all these formulas were designed for 2D coordinate systems and only dealt with the firing of a single shell. Also all problems were designed to cater for behavior of a single shell, while the gun works in the burst mode of firing.

4.4 Solutions.

In order to deal with these problems, the solutions which were calculated are presented in detail in paragraphs shortly.

4.4.1 2D to 3D Firing Solution.

No standard firing equation exists which can calculated the location of a single shell in 3D space. A tedious search of the relevant mathematical material resulted to no avail. However, a study of OpenGl functions reveled that the standard rotate and scale functions can help in displaying the single shell in any direction and at any location. However the problem was not over as still the location of shell was not known. The OpenGl function did not reveal the new location calculated by its internal function. It only displayed them finally on screen. However all behavior of the shells were successfully displayed. Finally the location of each shell was obtained by multiplying the 2d coordinate values with the rotation matrix. The same way OpenGl was doing to display the graphical results.

4.4.2 Burst Firing

The standard equations were only concerned with the firing of a single shell. With much effort burst effect was created. After its creation it was realized that a particle generator has been created. lots of different variables make a particle but normally used are Position -- Holds the world coordinates of the center of the particle, Velocity -- Specifies the direction and speed in which a particle is moving, Color -- A 32-bit color for the particle, Size -- The width and height of the particle quad, Life -- Length of time in seconds of how long a particle is active (i.e. visible), Angle -- An angle of rotation for the particle, A Texture –

defines shape of particle, and only points were used, Also all of particles are affected by gravity. The same gravity applies to all particles; other important things include a source, which in this case is gun. The life of particle is dependent upon factors like shell life time and the time of flight of particle. Ground hit also makes particle life zero. An array was used to hold all particles and all their variables. At each frame each particle is rendered as per the values of its variables.

4.5 The Shell Structure

The standard shell structure is based upon these variables

Float x; present x-axis location of shell in 3d space

Float z; present z-axis location of shell in 3d space

Float y; present y-axis height of shell in 3d space

Int type; shell types, in cindery, tracer, HE, etc

Float bearing; Direction in which shell was fired initially

Float elevation; Angle of Elevation in which shell was fired

Float speed; speed of shell....

Float start Time; Time at which shell was fired, all calculations should be time based

Int shell Status; 0 = in chamber, 1= in air, 2 = hit/destroyed, not moving any more.

With the help of all these variables, location of each shell is computed using the standard formulas mentioned earlier in 2d space. After which standard translation and rotation transformations are carried out using OpenGl calls.

Chapter 5

CALCULATING FLIGHT PATH OF AN UNMANNED BOMBER

5.1 Introduction

The dynamics of weapons technology are most apparent €in the field of air threats. In the next few years air defence will face enormous advances in the technical /tactical areas as they relate to tactical aircraft, attack helicopters, air-to ground missiles, drones, electronic warfare and most importantly and most dangerously the unmanned bombers. High performance avionics and improved armament, increased penetration capability and multifarious attack profiles make the Unmanned Bomber most furious.

Calculating flight path of an unmanned bomber is an active area of research in the modern era. Though it is a challenging job, but people have achieved lot of successes using different artificial intelligence techniques especially the Fuzzy Logic.

5.2 Complexity Factors

Complexity factors are those, which actually effect the calculation of flight path of an Unmanned Bomber. These are effect Of Target on Flight Path and Physical Shape of Target. Shape of target plays an important role in adopting the attack profile/techniques by the Unmanned Bomber. Main aim of an Unmanned Bomber or of any air craft is to achieve the required level of destruction of the target. If the objective is not achieved then all the efforts and technology is futile Targets like runways, bridges and railway stations require special

techniques to acquire the required level of destruction. Runways, for example, require air craft to come from a direction which is along the axis of the runway. If air craft attacks the runway from a direction perpendicular to the direction of the runway, then all the planning and the effort made is useless. However this calculation is done by the sender of unmanned weapon, and till now no technique has been developed to calculate the effect at run time.

5.2.1 Target Nomenclature

Target nomenclature is also one of the important factors for adopting the attack techniques. Petroleum and Lubricants Depot, Ammunition Factory, Nuclear Installations are some of the targets which require special attack techniques. When Petroleum and Lubricant Depot is to be attacked, aircraft has to come at a high altitude to maintain its own safety. Further if a Nuclear Installation is to be attacked aircraft has to attack from an even higher altitude. If it makes a lower attack profile to attack such targets, ultimately it will also be damaged due to massive destruction of the target.

5.2.2 Moving or Stationary Targets

In type of targets, Moving or Stationary targets have their own role in dictating the type of weapon to be used and the flight path to be followed .Moving targets like vehicles require the flight path to be in the direction of move of the vehicles. If the flight path is in the perpendicular direction to the direction of move of vehicles convoy then the probability of achieving the maximum destruction become very less. Further if the weapons are released from a very high altitude, then there is a chance that a fast moving target may get out of the dangerous zone of the weapon resulting in reducing the hit probability and missing the

objective. Targets like persons may require that a high altitude bombing is carried out. Since diving down and delivering the rockets at a scattered human population may result in wastage of ammunition and aircraft fuel. Targets like tanks, most of the times, require a low level attack techniques .Since tanks have anti air craft guns mounted on them, the attack should be carried out with great care. 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 field like Artillery concentration area and deployment areas have a specific shape. While attacking the stationary targets like Artillery areas, air craft is bound to attack from a specific direction to achieve maximum destruction of the target. If the attack is carried out from an unplanned random direction, probability of achieving the maximum destruction is minimized.

5.2.3 Direction of sun

Direction of sun plays an important role in dictating the flight path of any aerial vehicle may it be an Unmanned Bomber or an air craft. An attack coming from the sun, or going into sun after attack, is bound to make things more difficult for the defenders. Without radars, detection of aircraft is very difficult.

5.2.4 Type of weapons

Type of weapons has a 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 is as follows.

5.2.4.1 Retarded Bombs

For the delivery of Retarded Bombs, initially the air craft has to fly at very low altitude of approximately 50 meters. When the aircraft is at an approximate distance of 1000 meters, it pops up and releases its weapons. After it has released its weapons on the target it again pops up and flies back.

5.2.4.2 Cluster 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

5.2.4.3 Dispenser

For the delivery of Dispensers, the air craft has to fly at a very low altitude of approximately 40 meters .In case of Dispensers, the air craft releases its weapons just after reaching the target area, as it is flying very low. After it has released its weapons on the target, it flies back maintaining the same altitude.

5.2.4.4 Napalm Bombs

For the delivery of Napalm bombs, initially the air craft has to fly at a very altitude of approximately 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.

5.2.4.5 Rockets

When the mounted weapons are rockets, the aircraft comes at low altitude to attack the target. When the air craft is at an approximate

distance of 3000 meters from the target, it pops up and aligns its direction with the target making an angle of 20 degrees with the horizontal and start diving down towards the target very smoothly. When the air craft is at an approximate distance of 2000 meters or at a time distance of approximate 2 seconds, which may vary with speed of Bomber, it delivers the weapons. Just after delivering the weapon it pops up and moves back to its take off place.

5.2.4.6 Guns

When the mounted weapons are Guns, the aircraft again comes at low altitude to attack the target. When the air craft is at an approximate distance of 2500 meters from the target, it pops up and aligns its direction with the target making an angle of 10 degrees with the horizontal and start diving down towards the target very smoothly. When the air craft is at an approximate distance of 1700 meters or at a time distance of approximate 3 seconds (since while firing guns speed of air craft has to be a bit slow then while firing rockets) , which may vary with speed of Bomber, it delivers the weapons for the achievement of its destined objectives. Just after delivering the weapon it pops up and flies back.

5.2.4.7 Anti-Radiation Missiles

When the weapons are Anti-Radiation 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 target, it pops up and aligns its direction with the target making an angle of 20 degrees with the horizontal and start diving down towards the target. When the air craft is at an approximate distance of 1000 meters, it

delivers the missiles for the destruction of its destined objective. Just after delivering the weapon it pops up and flies back.

5.2.4.8 Ballistic Bombs

When the weapons are Ballistic Bombs, the aircraft again comes at an altitude of 1500 meters to attack the target. When the air craft is at an approximate distance of 3000 meters from the target, it pops up and aligns its direction with the target making an angle of 30 degrees with the horizontal and start diving down towards the target. When the air craft is at an approximate distance of 1700 meters from the target, it delivers the weapons for achievement of its objective. Just after delivering the weapon it pops up and flies back.

5.2.6 Terrain features

Terrain features has their own role in dictating the flight path of Unmanned Bomber. In mountainous terrain, the Bomber has to move with the terrain features else it will hit some terrain feature and will be destroyed before achieving its objectives. Similarly, while passing by a thick jungle, a low flying Bomber has to adjust its flight path to avoid collision with tree tops. Plane grounds also affect the flight path of Bomber. 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.

5.2.6 Fire of Air Defence Weapons

Fire of Air Defence Weapons has a direct bearing on the flight path of a Bomber. If the target area has the weapons with larger ranges, Bomber may not get much closer or may not come at low altitude to engage the target. Similarly, if the weapons deployed at the target area have small

ranges, the Bomber may get very close to target and may engage it with more concentration.

5.3 Chosen Complexity factor

Out of all above complexity factors, the factors chosen to be incorporated in the project were Type of weapon, Type of target and Terrain Features. Incorporation of other factors depends on the availability of time, and may be done later upon demand from the user organization.

5.4 Implementation Details

The main problems encountered were accurate shape, behaviour, frame rate etc. The details are discussed briefly.

5.4.1 Problem Areas

The problems faced in simulating behaviour of an unmanned bomber are:

5.4.1.1 Accurate Shape

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.

5.4.1.2 Accurate Behaviour

An aircraft moves with a specific speed, and its movement is also well defined in terms of angle of turning, elevation angles, rate of turning and climbing etc.

5.4.1.3 Differing Computer capabilities

Keeping in view the speed of processor, RAM etc, the same simulation may run with varying speeds on different machines. This problem unless addressed poses a big problem which hinders in creating true representation of accurate flight paths. Each system can show the path being traversed at a different time rate.

5.4.2 Implementation Progress

The steps necessary in order to create a true representation of the flight path of bomber comparable with real world are, model made available in 3D space, able to be placed anywhere. Basic movements of a stationary model, i.e. heading, pitch, yaw, roll etc. Decided units of work compatible with real world coordinates, in meters vs. 0.1 glUnits .Creating a universal frame work for all types of paths to be traversed at a constant rate on all types of different machines, using system clock ticks, and fps based calculations. Time based movement achieved .Achieved movement in 2D plane, with fixed height. Implemented movement of aircraft in any direction in 2D plane. Decided framework for predefined path of fixed aircraft movement .Achieved degree of fuzziness with reducing the number of control points given as inputs. Planned behaviour includes only start point (random) and location of target (moving or stationary) given as input. Rest all flight paths will be calculated by the fuzzy flight path calculator of bomber.

General attack profile characteristics of different weapons will be fed as guideline in calculating flight path. Aircraft will follow the path with accurate heading, elevation angles, attack angles and accurate speeds, using fuzzy logic.

Chapter 6

Target Tracking and Results

6.1 Introduction

A simulator is of no use if results showing the overall performance of a training session are not displayed. Important training parameters include things like the variation from the standards, etc. The results help us make and train better soldiers and get the maximum out of us by competing and pitting us against each other and against ourselves. The Oerlikon simulator thus needed to be equipped with a results screen showing the parameters and out come of the training session. Also the Tracking is carried out; however the interpretation is as per comprehension of the problem which will be discussed shortly.

6.2 The Oerlikon Tracking Behaviour

The current position of the target is determined by tracking. This occurs in two phases which are two dimensional tracking and three dimensional tracking.

The determination of the future position of the target at the hitting point results from calculations based on the current position of the target, its speed and acceleration, together with various ballistic factors ballistic factors as shown in Fig 6.1.. This involves Calculation of the lead angle

Figure-6.1 Bearing, Elevation and Distance

6.2.1 Two Dimensional Tracking

By keeping the target in the centre circle of the periscope reticules the azimuth and elevation angles can be measured by the mirror coders and sent to the computer.

Laser range finding is initiated as soon as the TRACKING button is pressed. It can take a little time, however, before the range finding process can produce reliable results. During this time, therefore, the slant range to the target is still only an initial value or an estimate.

This case, where the actual distance is unknown, is called two dimensional tracking. Only two of the three elements required for the determination of the position of the target - the angular values - are known with any accuracy.

6.2.2 Three Dimensional Tracking

Three dimensional tracking means that the laser rangefinder has measured the actual slant range to the target. A laser pulse is emitted every 240 ms (approx. four times a second) during tracking and the computer therefore also receives distance measurements at this rate.

The three elements required to determine the current position of the target are therefore now known.

6.3 The Problems

The general tracking behaviour of gun was known to us, however to implement the same in computer, and most important of all to simulate it correctly was the main challenge. The main problem was that of to calculate the lead angle of shells to hit the target accurately, which consisted of many complexity factors.

6.3.1 The Lead Angle Complexity Factors

The factors are The Direction of Movement of Aircraft, Crosser Aircraft, Air craft Approaching towards gun, Air craft moving away

from gun, The Acceleration or Deceleration of Aircraft, and finally Aircraft moving in a smooth arc, combined with the above two factors.

6.3.2 The Frame Rate

The Frame rate based movement of Air craft, providing accurate visual speed to the trainee. This was a must limitation; other wise wrong training to the troops would have been given.

6.4 The Solution

The last mentioned problem was the most important, as it made any solution to the lead angle solutions impossible to implement. Consider this, in any case, if the movement of aircraft is frame rate based, at each frame the aircraft will take a big jump in 3d space to make up for the visual accuracy. For example, if an aircraft is moving at a rate of 300 meters per second, and the current frame rate is 60(considered very well) then the air craft has to move 20 meters in each frame to show visual accuracy. This means that even if the lead angle is calculated accurately, the shells may not hit the air craft because the hitting point lay in the space between 2 frames.

6.4.1 Occam's razor

Named after a famous scientist, the postulate says that "The simplest solution is the best solution". So keeping to the suggestion, A solution was found in the tracking and firing behaviour of gun itself. The gun is 99% accurate, provided all parameters are correct, so it means that if the tracking is being carried out accurately, any burst's hit probability is 99%. Relying on this single fact a modification was made in tracking and hitting solution to correct tracking. Now if the gunner is carrying out 3D tracking and fires a burst, the aircraft will be hit by the burst. Sample result Screen is shown in Figure 6.2.

6.5 Results Parameters

Important Results parameters include Deviation from standard, Deviation in Elevation, Deviation in Bearing, Total Practice Time, Hit / Firing Efficiency, Total Round Fired, Total Round Hit, Hit Percentage, Tracking distances during engagement, Efficiency of tracking Types, Time of Target out of sight, Time of target in sight, Time of target 2D tracking, Time of Target 3D tracking, All these parameters are being calculated in a compact form on a single screen, giving a very compact output in a small space. Colour lines giving the intended tracking mode.

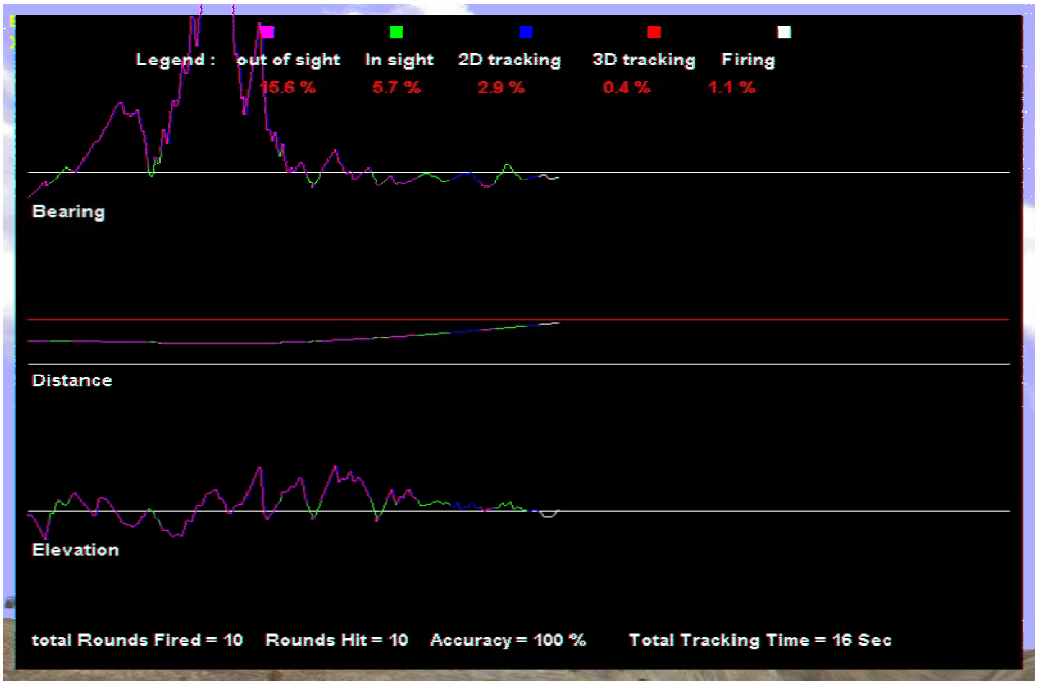

Figure-6.2 The sample results screen

6.6 The Result Structure.

The results data is stored in a structure of a total of 1000 entries. Which include Tracking Data, Float acElev, gives elevation of aircraft from own location, Float acBg; gives bearing of aircraft from own location, Float OwnElev; present elevation of gun in world coordinates , Float OwnBg; present bearing of gun in world coordinates , Int

tracking Type; tracking, type 2d, 3d, in sight, out of sight, Float distance; the distance of aircraft from the gun, Bool fire; whether firing or not.

Chapter 7

Resource Creation

7.1 3DS

3DS is a 3d file format, which can be viewed using OpenGl. 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.

7.1.1 Advantages of using 3DS files

A very popular format, 3DSMax, which seems to be quite popular among artists, exports to this format. Unknown parts in the file can be skipped easily. File easy to read. Optional per-vertex texture coordinates .

7.1.2 Disadvantages of this format

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 accuTrans and Maya.

7.2 Tools

7.2.1 Accu Trans 3D

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. Textures and UV

coordinates are supported. The program has been enhanced with additional features to make it more than just a 3D file conversion program.

7.2.2 Accu Trans 3D Files Conversion Capability

AccuTrans can successfully translate between many file formats mentioned in table 7.1.

Table-7.1 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

7.2.3 Other Features

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). Align surface normals for 3D Studio, LightWave, Lightscape, RenderWare and VRML. Select and deselect the layers to be saved to a file. Scale 3d objects when either reading or writing files. Create planar, cylindrical or spherical UV texture coordinates.

7.3 TerraGen

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

7.3.1 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), modify it in various ways or perform arithmetic with other terrains. The surface map can also be created.

7.3.2 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

7.3.3 Terrain Genesis

7.3.3.1 Method

The method used to generate the terrain. The Subdivide & Displace method is the 'original' generation method. Ridged Perlin is an extension of Perlin Noise, and creates landscapes with more ridges (!). There is also Multi- versions of Perlin and Ridged Perlin now available which seem to generate more craggy, irregular landscapes. The best way is to experiment until appropriate results are obtained.

7.3.3.2 Action

If there is a requirement to Erase First and generate a new terrain. This starts from scratch with a new terrain. Generating features on the existing terrain generates a new random terrain and combines it with the existing terrain.

7.3.3.3 Realism

A higher setting generates a more realistic terrain with smoother transition between high and low. A lower setting can be used to create a more "craggy" landscape. Only available for the Subdivide & Displace method.

7.3.3.4 Smoothing

Adjusts how smooth the landscape will be. Setting this too high can interfere with the realism setting and create unwanted results. Only available for the Subdivide & Displace method.

7.3.3.5 Glaciations

Modifies the landscape by flattening valley bottoms and smoothing sharp changes in gradient.

7.3.3.6 Canyonism

Sharpens the valley bottoms, creating an effect similar to canyons in the desert (although usually not quite so pronounced). In many ways, Glaciation and Canyonism work against each other. Applying both during the creation process can create a more realistic, balanced landscape as found in nature. If one wish to create an image of desert canyons one might wish to set Glaciation low, and canyons high, etc.

7.3.3.7 Size of Features

This slider controls how large the hills are, both in horizontal and vertical size.

7.3.3.8 Perlin Origin

Used in the generation of Perlin landscapes (and the Perlin variants). the "random origin" feature is used unless there is requirement to reproduce an earlier result. By changing these values slightly, one can move the landscape in small increments, which is useful in creating a landscape with nice features which are partly out of view.

7.4 Height Map

A height map is a set of Gray scale colour values that determine "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. To tile a second texture on top of the first one to give the appearance of more detail. This is called detail texturing, which can add a great deal of realism to a scene. Multitexturing is used to achieve this neat effect.

First, 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 is generally wanted to be used because to either

know what the size and type are, or guess. GL_TRIANGLE_STIP was used. 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 the same way. If not done this way, one will get polygons stretching across the whole terrain. Multitexturing was added so that a detail texture could be applied over terrain. This gives the terrain a more detailed look, instead of a stretched look. To do this, normal Multitexturing functions was used, and detail texture's properties were changed with GL_COMBINE_ARB and GL_RGB_SCALE_ARB. These 2 flags allow us to increase the gamma on the detail texture so that it doesn't over power the texture of the terrain. The last thing fiddled with was 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. When the space bar is hit, this changes the scale value to get a different tiling effect. It eventually wraps around again.

7.5 XFrog Trees

7.5.1 Horse Chestnut

Tree, deciduous broadleaf Shape: broadly columnar
Origin: Located southeast of Europe (Albania, Northern Greece)
Xfrog models: 30 m., 15 m., 4 m.
Environment: mountain woods, up to 1.300m.
Climate: mild, temperate
Notes: often used for urban decoration because of the beautiful shape, springtime blossoms and dense summer shadows. The name derives from an old Turkish habit of grinding the seeds for use as curative

food for winded horses. One of the most popular trees for decorating city boulevards.

7.5.2 Banana

Plant Origin: Asia

Xfrog models: 2 to 5 m.

Environment: tropical valleys, in full light to light shade

Climate: warm and humid, mild

Notes: bananas are the world's 4th largest fruit crop today. The Banana plant makes fruits only in a tropical environment; it can live in mild climates, but there it almost never makes fruits. Although the plant thrives in full sunlight, the fruits are best kept if the plant is in a light shade. Banana plant needs protection from the winds.

7.5.3 Canary Date Palm

Xfrog models: 12 to 19 m.

Environment: coastal forests

Climate: warm, mild

Notes: tall, beautiful decorative palm that can grow up to 20 m.

The Latin name indicates not only the native region, but also the fact that Phoenicians first made this palm known to the ancient Greeks. In fact, the Canary Date Palm diffusion in the Mediterranean area dates back to the Phoenician age.

7.5.4 Organ Pipe Cactus

Plant Origin: Located southwest of USA, Northern Mexico

Xfrog models: Lemaireocereus thurberi: 4 m.; green 4,7 m.; green 4,1 m.; green

Stenocereus thurberi: 3,5 m.; green 4,2 m.; green 3,7 m.; green

Environment: arid areas

Climate: hot, warm

Notes: the Organ Pipe Cactus is a large cactus forming a cluster of stems up to 6 m. tall. Several similar species exist in the native

regions. Blooming is June through July. The fruit is edible and harvested by native areas people. The fruits can be stewed into jam or candied.

7.5.5 Pencil Cholla

Plant Origin: Located southwest of USA, Northern Mexico
Xfrog models: 1,6 m. 1,4 m. 1,2 m.
Environment: arid areas, in full sunlight, on sandy very well-drained soils.
Climate: hot, warm, mild
Notes: also known as “Diamond Cholla” or “Branching Pencil”. It blooms in late Spring. The numerous minute spines are easily dislodged at the simple touch, and they get stuck in the skin, where they
are difficult to remove from. The Pencil Cholla is fairly cold-tolerant.

7.6 Clouds

Clouds are created in Paint Shop Pro using the masking technique. A simple procedure is explained in the next heading. The result of cloud rendering is shown in figure 7.1.

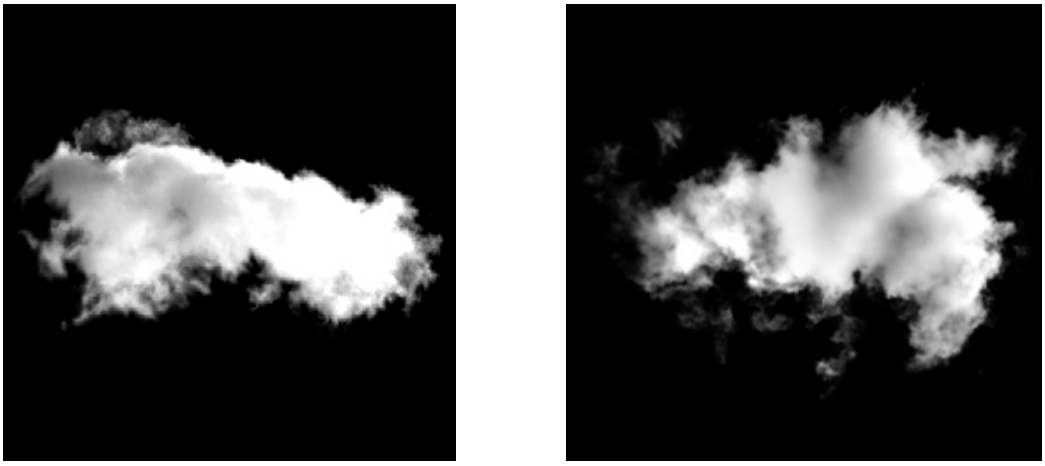

Figure-7.1 Sample Clouds

7.7 Creating Masked TGA in Paint Shop Pro

7.7.1 Creating TGA Files

A step-by-step tutorial to creating TGA graphics files for use with the transparent graphics will be presented here. It is based on Paint Shop Pro but Photoshop users shouldn't have much difficulty substituting Photoshop commands for those used here.

7.7.2 Stage 1

Click on the "File" menu then on "New" to bring up the box as shown in Figure 7.2, change the size to Width 127 and Height 127.

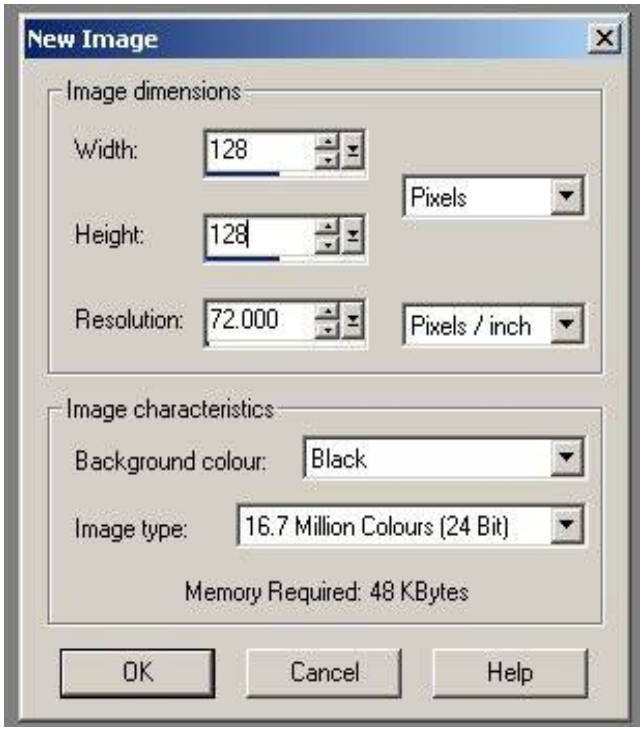

Figure7.2 Step 1 Dialog

7.7.3 Stage 2

This will open a new window into graphics can be drawn a suitable font have been used to draw the number 6 in white with a near-black color outline as shown in Figure 7.3.

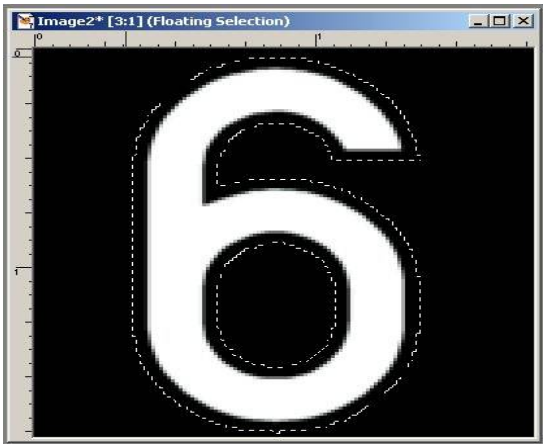

Figure-7.3 Inverted Selection Using Magic Wand Tool

7.7.4 Stage 3

When drawing the graphic have been finished, Select the graphic and

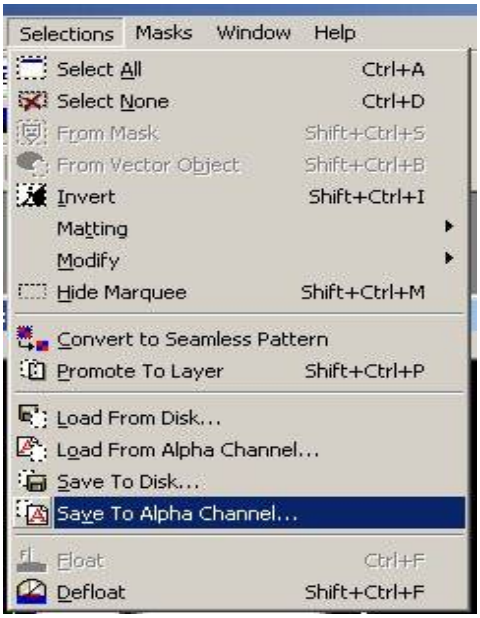

Figure-7.4 Save to Alpha Channel

click on the "Selections" menu to bring up the drop down box as shown in as shown in Figure 7.4, and select the "Save to Alpha Channel" option.

7.7.5 Stage 4

This will display the box in fig 7.5. In this we save the available alpha channel in the TGA.

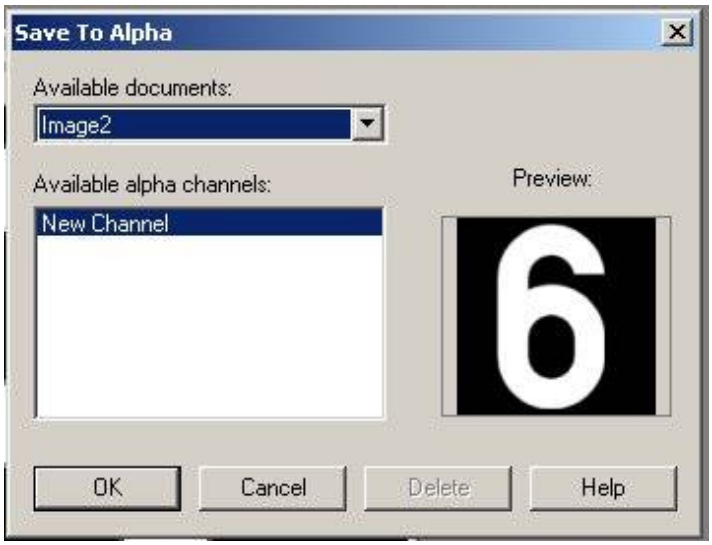

Figure-7.5 Available Channels

7.7.6 Stage 5

When the next box shown in figure 7.6 appears click on OK again.

Figure-7.6 Alpha Channel Name Selection

7.7.7 Stage 6

Click on the "File" menu and then on "Save As" shown in figure 7.7

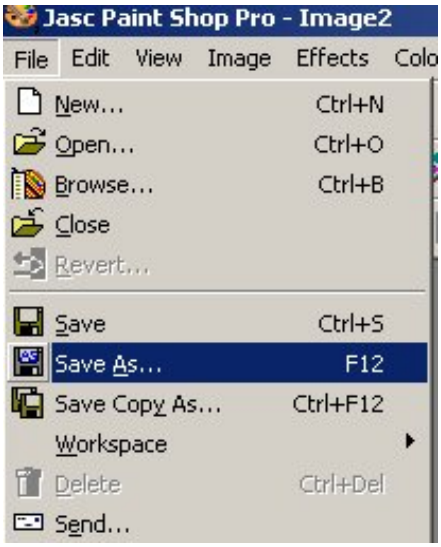

Figure-7.7 Save As TGA

7.7.8 Stage 7

Give the file a name, e.g. "White6" and make sure that the file type is Targa set to "True vision" (.tga files), as shown in Figure 7.8.

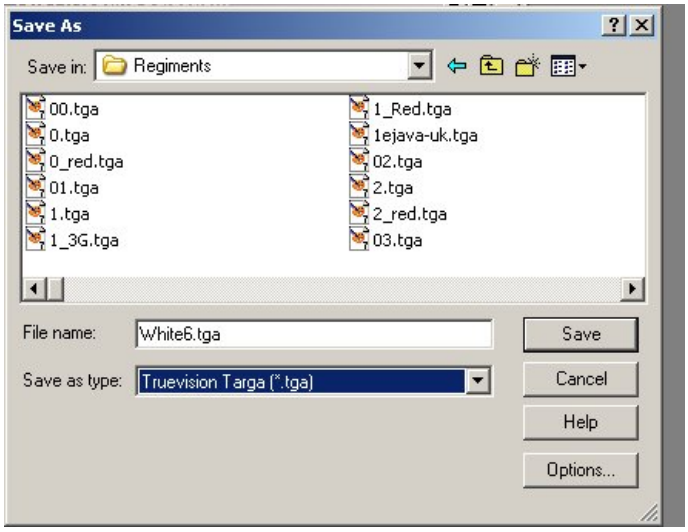

Figure-7.8 Save As TGA Dialog

7.7.9 Stage 8

Before saving the file click on the "Options" setting to bring up the box and make sure that the Bit depth and Compression settings are set as per the example shown in Figure 7.9,.

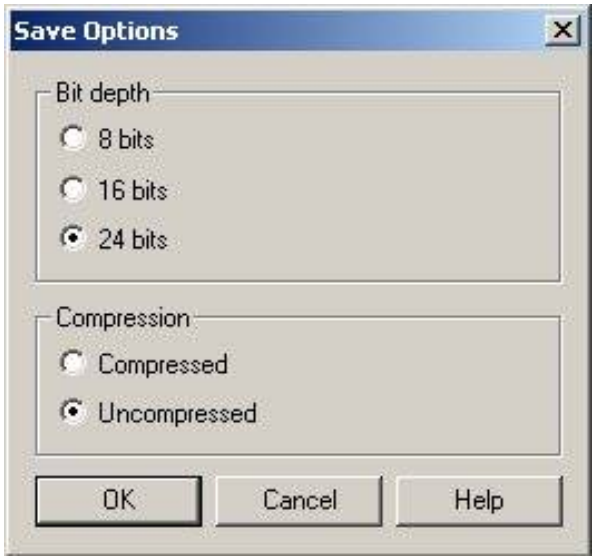

Figure-7.9 TGA Options

Lastly click on "OK" to save the file. The most important thing to note is the "Uncompressed" option which is essential to display the Alpha information. In my example the area surrounding the graphic is black but it can be any color as long as the bottom left pixel contains the desired transparency color as this is what is used to determine what areas should be transparent.

Chapter 8

Gun Menu System

8.1 Overview

The operation of the individual subsystems is dealt with in the appropriate chapters of the EDO. The task-related operation of the gun is described in the Drill book. This chapter describes the menu programs used in the GDF-005 gun. The menus are described in two main categories according to their function. Operation Menus are used in bringing the gun to firing readiness and Technical Menus are aids for carrying out technical checks of the gun

8.2 Use of Menus

Menus are computer programs. They guide the operator step by step through procedures for setting up the gun. They enable the gun to be brought to firing readiness in a controlled and systematic way and provide a means of viewing and checking gun data after each firing break. The communication between the operator and the computer takes place using the display and the keypad. Only numerical values can be keyed in. All horizontal angles are entered in the form of angles to north (azimuth angles). This means that the computer has to be given the alignment of the gun in relation to north. All elevation angles are entered with reference to the lower mounting. A mistake or the entry of a value outside the range permitted for the entry causes all or part of the word ERROR to be shown in the display. The amount of the word ERROR that appears on the screen depends on the size of the entry expected, that is, for a two character entry such as a menu number only the first two characters of the word ERROR would be displayed .In the same way, a three character entry would display only the first three characters .

Display SALVOTIM ERR SEC

A menu is started by pressing the RET key. Menus 1 and 8 run automatically after pressing the RET key. Menus 2, 3, 4 and 10 are divided into subprograms. At the beginning of a subprogram a question appears in the display, e.g. ALIG_CHECK?. Pressing the YES key starts the subprogram; pressing the RET key skips over the subprogram. Menus 5, 6 and 7 ask the question CHANGE DATA?. If this is answered with the YES key the data that follows can be overwritten. By answering the question with the RET key the data will be displayed for checking but cannot be changed. In subprograms which are used to set and store one or more gun positions the gun is moved using the hand drives. Subprograms in which gun positions are checked, however, require the electrical drives to be activated.

8.2.1 Keypad Operation

The keys on the keypad have many functions:

8.2.1.1 Number Keys 0 to 9

These keys are used to enter menu numbers during menu selection and numerical values within menus.

8.2.1.2 Decimal Key

This key allows decimal values to be entered, for example when entering azimuth and elevation values to an accuracy of one decimal point.

8.2.1.3 Minus Key

This key enables negative values to be entered, for example: air temperature -10'C

8.2.1.4 RET Key

This key has two functions, working in a menu step by step and as a NO answer to program questions

8.2.1.5 DEL Key

This key allows typing mistakes to be corrected, one character for each press of the key.

8.2.1.6 YES Key

This key answers program questions with YES, or in affirmative.

8.2.1.7 END Key

This key stops or ends a menu program CAUTION: Menu 3 and 4 can only be terminated after storing at least 3 points.

8.2.3 General Instructions for Menus 1 to 10

The description is written for the user and not the system programmer; particular menus are selected as an aid in carrying out specific tasks. Those menus which are used in several applications are described several times, but each time in relation to the task in question. Although the menu descriptions are arranged in the sequence 1 to 10, in practice the menus are normally carried out together:

8.2.3.1 Menu 1

Drift trim after switching on the gun drives, that is, after the POWER switch has been put to SERVO.

8.2.3.2 Menu 2, 3, and 4

The north alignment of the gun and positioning and sighting on procedures are carried out with most accuracy using the hand drives. As mentioned previously, checking gun positions can only be carried out when the electrical drives are activated. This means that it is best to carry out the entire setup procedures one after the other using the

hand drives and then to carry out the checks together after the electrical drives have been activated. It is important to realise that the ballistic lead is only switched off in Menu 2, and that therefore only in this menu is the barrel axis parallel to the sight (mirror) axis. If a horizontal safety margin is needed the table should be used to find the extra safety angle required. The table considers only the cases of the effective combat distance of the system (4000 m) and the maximum range of a practice round fired at an elevation of 700 mils (12000 m).

Speed of cross wind (m/s)	10	20	30	40
Effective distance 4000 m	6%.	12%.	18%.	24%.
Maximum distance 12000 m	20%.	40%.	60%.	80%.

8.2.3.3 Menu 5 and 6

The firing data are best entered after the gun has been initially loaded. Menu 6 is also used, however, after each reloading of the gun, as well as after unloading.

8.2.4 Symbols used in Menu Charts

Information in the display .Text in brackets, in a light typeface and written in upper and lower case represents a variable value, for example (Azimuth) = azimuth value (as displayed by the program or entered by the operator)

Input muzzle velocity (VO)

750 .. 1350 m/s

8.2.5 Starting the Menu System

The gun must be in the firing position and the electrical switch positions should be as described in Electrical Gun Drive. On the control box: POWER switch at ON TROUBLE lamp flashes On the operation panel GUN KING....appears in the display, followed by CODER NOT CALIB the first display shows the program version,

represented here by the subsequent display indicates that the coders are not calibrated and that the gun has no internal reference. The gun is not ready for operation. The coders are automatically calibrated as soon as: On the control box POWER switch at SERVO TROUBLE lamp goes out READY lamp lights up On the operation panel MENU 1 appears in the display The system is ready to be used with the menus. If the gun drives cannot be activated the gun can be calibrated using the hand drives. The gun is moved simultaneously in traverse and elevation by hand until MENU 1 appears in the display.

8.3 Operation Menus

Ten menus are available for use in bringing the gun to firing readiness. They appear in the display on the operation panel in ascending order. After Menu 10 has been completed Menu 1 appears again in the display. The operation menus are Menu 1 Drift Trim, Menu 2 Alignment, Menu 3 Laser Sector, Menu 4 Fire Sector, Menu 5 Distances, Menu 6 Firing Data, Menu 7 Meteo Data, Menu 8 Quicktest, Menu 9 Error Codes, and Menu 10 Servo Sector.

8.3.1 Menu 1 DRIFT TRIM

Menu 1 is used to automatically define the zero position of the control yoke according to the error voltage. Menu 1 is carried out every time the gun is switched on. The procedure is to Select Menu 1, Press RET to start menu. DRIFT TRIM appears in the display warning is that the gun moves slightly in traverse and elevation**.** The program ends automatically. MENU 2 appears in the display if the gun still drifts in traverse and/or elevation in spite of Menu 1 being carried out, repeat Menu 1.

8.3.2 Menu 2 ALIGNMENT

The gun is aligned after emplacement (Set Gun Alignment).The

correctness of the alignment is checked after being set and at suitable opportunities (Check Alignment).The gun is also aligned with the Fire Control Unit, either directly or indirectly (Set Alignment to FCU).The retro-reflector has to be mounted on the gun during the alignment procedures. Note that Ballistic and dynamic lead is suppressed in the subprograms of this menu.

8.3.2.1 Set Gun Alignment

This procedure aligns the azimuth zero point of the gun with north. Since all horizontal angles are entered in mils with reference to north (that is, as azimuth angles) the first step in bringing the gun to firing readiness must be the determination of the direction of north. This can be done using a theodolite or a map reference. The theodolite is set up at the centre of the fire unit in order to be able to align the FCU as well as both guns with north. The theodolite scale is set at 3200 mils towards north. The gun being aligned and the theodolite sight on each other. On the gun, using Menu 2, the position of the theodolite is stored and the angle measured by the theodolite is keyed in. Azimuth 0 now corresponds to north. Menu 2 is used to store the position of the terrain point and the angle read off the map is keyed in. Azimuth 0 now corresponds to north. Two fixed points are also defined: Fixed Point 1: For direct alignment Fixed point 1 is the FCU. for indirect alignment he theodolite. This point is used later to permit a rapid alignment with the FCU. Fixed Point 2: This is a terrain point chosen for subsequent use to check whether the gun is still aligned and/or level. The point chosen should have the characteristics:
Easily defined in traverse and elevation Unmoving Illuminated/visible at night The position of this point is stored during the setup procedure. In the checking procedure the gun aims at this point and enables the alignment and level to be checked for consistent accuracy. Finally, the position that allows the optimal loading and reloading of the gun is determined and stored.

8.3.3 Menu 3 LASER SECTOR

The laser beam of the rangefinder is dangerous and for this reason its use can be restricted to a predefined laser sector. A laser sector is the area in which the laser can be triggered. Outside this sector the computer disables the laser. The sector is defined by storing points on its boundary. The menu is divided into two subprograms: The first subprogram allows the stored points of the sector to be checked. This is done after firing breaks. The check is only possible when the electrical drives are activated. The check can be ended at any time by pressing the END key. The second subprogram allows the points of the sector to be stored. In this subprogram the gun must be moved using the hand drives. The laser sector affects both the electrical initiation (TRACKING button) and the mechanical initiation (laser/trigger pedal) of the laser. The sector boundary is defined by a minimum of 3 and a maximum of 20 points lying within the movement range of the gun. Only after having stored at least 3 points is it possible to leave the menu. The computer numbers the points in ascending order and links them together along the shortest path. The first point stored is linked to the last point stored and the sector closed. The lines between the points form the sector boundary. In practice, the three types of laser sector are of importance. No limitation: The laser sector is identical with the entire movement range of the gun. This results when three points are stored separated by 2133 mils (120') and at a minimum elevation. This is best achieved in practice by positioning the barrels in full depression over each of the three jacks in turn and storing these points. Terrain limitation: Terrain limitation is set when the laser must not be used in terrain within the laser safety radius. It is possible, using between 3 and 20 points, to define a sector which does not restrict the use of the gun against air targets. A terrain limitation is produced when

the angle between the first and the last point stored is greater than 3200 mils and the circle is thus closed along its shortest path. The laser is disabled when pointing below the defined boundary. Laser window: A laser window is produced when at least four points having an azimuth angle difference of less than 3200 mils are stored. This restrictive limitation is used on the firing range and offers the highest level of safety.

8.3.4 Menu 4 FIRE SECTOR

For safety reasons it is often necessary to limit the area in which the gun can be electrically fired, for example to restrict particular areas of terrain. Menu 4 is used to set a fire sector. The structure of the menu is identical with that of Menu 3, Laser Sector and its operation is also the same. The step by step descriptions are repeated in full for the sake of completeness. A fire sector is the area in which the weapons can be fired with the electrical triggers (when all other safety conditions have been fulfilled). Outside this sector the computer disables the electrical triggers. The fire sector does not affect the manual trigger (laser/fire pedal). The gun can be fired manually in any direction. The menu is divided into two subprograms: The first subprogram allows the stored points of the sector to be checked. This is done after firing breaks. The check is only possible when the electrical drives are activated. The check can be ended at any time by pressing the END key. The second subprogram allows the points of the sector to be stored. In this subprogram the gun must be moved using the hand drives. The sector boundary is defined by a minimum of 3 and a maximum of 20 points lying within the movement range of the gun. Only after having stored at least 3 points is it possible to leave the menu (END key).The computer numbers the points in ascending order and links them together along the shortest path. The first point stored is

linked to the last point stored and the sector closed. The lines between the points form the sector boundary.

8.3.5 Menu 5 DISTANCES

This menu is used to measure laser distances in the terrain and to store two fixed distances under SHORT and LONG. The menu consists of two subprograms: One subprogram enables distances to be measured with the laser and stored. The other subprogram allows the stored values to be checked. The subprogram required is chosen at the question CHANGE DATA? The distances stored here as SHORT and LONG are used as the two fixed distances selected at the LASER RANGE switch on the operation panel. When the gun is operated without the laser rangefinder being used one of these switch positions is selected. The choice of the fixed distances is based on tactical considerations, and taking into account whether engagement will take place against ground and/or air targets. Possible applications are: Determination and entry of the firing point for ground targets initially outside the laser sector. Determination of the first and last firing point for air targets in the primary sector.

8.3.6 Menu 6 FIRING DATA and ACCOUNT

Menu 6 is divided into three sections:

8.3.6.1 Firing data

The first part of the menu consists of two subprograms; one subprogram enables the firing data to be set, the other is used to check the stored values. The subprogram required is chosen at the question CHANGE_DATA?

8.3.6.2 Ammunition account

The second part of the menu allows an ammunition account to be

kept. The number of rounds fired per barrel is recorded by counters. These counters can be read out or reset. The ammunition reserve can be entered and is automatically updated during firing by the round counters. The current ammunition reserve level can be read out at any time.

8.3.6.2 Rates of fire

The last part of the menu enables the rate of fire of each cannon to be read out. Menu 6 is used in the applications like, Input firing data, Check firing data, Ammunition accounting, after loading the gun, after reloading the gun, after unloading the gun, Check rates of fire checking the rates of fire of the two cannon is best carried out after reloading. At this point several rounds have been fired and the appropriate part of Menu 6 has already been reached after the ammunition account has been updated

8.3.7 Menu 7 METEO DATA

The menu consists of two subprograms. One subprogram enables meteorological data used by the computer in the calculation of lead angles to be entered; the other is used to check the stored values. The subprogram required is chosen at the question CHANGE DATA? If the question is answered with YES the meteorological values can be entered. Pressing RET allows the values to be checked. It is not possible to correct the stored values during the check. The gun is not equipped to determine this meteorological information. The values should be obtained from the FCU .If the gun is being operated autonomously (that is, without the FCU) or the FCU crew has not yet determined the meteorological values the procedure adopted states that The air temperature can be measured using a whirling thermometer and the air pressure at ground level can be determined with a barometer. These values at least should always be entered. The wind speed and direction should be entered as 0. Any other

entry, particularly as far as the wind direction is concerned, could lead to a cumulative error. The wind at ground level can also be determined quite accurately using an anemometer and compass. If the height between the measured air pressure position (measuring point) and gun position exceeds 30 ft (10m) the barometric pressure to be entered into the gun computer must be compensated for.

8.3.8 Menu 8 QUICKTEST

Menu 8 allows the significant parts of the logic circuits, gun drive control system and the sight control system to be tested automatically. If a fault is detected the TROUBLE lamp on the control box blinks. While the test is in progress do not change any switch positions. These actions can lead to incorrect error reports. The preconditions for the correct performance of the test are: The readiness check must have been carried out in order to ensure that all switch positions are correct. The gun moves erratically in traverse and elevation while the test is in progress.

8.3.9 Menu 9 ERROR CODES

In the Simulator this part is not implemented as this the work of technical persons to deal and not the operator and trainer .So only Menu 9 appears with message that no errors .Menu 9 allows the error codes to be read out. The presence of such codes is indicated by the flashing of the TROUBLE lamp on the control box. The TROUBLE lamp is activated by the built-in test circuits as soon as a fault condition is detected. The Drill book contains a list of these error codes together with instructions for subsequent action.

8.3.10 Menu 10 SERVO SECTOR

Menu 10 enables movement limits to be set for the traverse and elevation servo motors. The limits are entered as azimuth and elevation

angles and form a servo sector. The limits of the servo sector are purely electronic, that is, using the hand drives the gun can be physically moved outside the servo sector at any time. The servo sector is normally used on the firing range. In this case a servo sector equal to or smaller than the firing range sector is defined. The servo sector as a whole can be enabled or disabled at the beginning of Menu 10:

Chapter 9

Analysis and Results

9.1 Introduction.

The most important part of any project is the results derived from it. The Oerlikon Simulator Project being a large project in its size and level of effort also leads to many open ended results and questions. The areas covered are many, and in this chapter they will be discussing them one by one.

9.2 Terrain Engine Analysis

The terrain engine being the most visual part of this project provides an intriguing an engaging view to an open terrain. However there are aspects which need more attention and can be even further improved.

9.2.1 Infinite Terrain

With some effort the visual aspect of infinite terrain can be added, making the terrain engine even more realistic. This can be done using two techniques.

9.2.1.1 DEM

The original DEM or digital elevation maps of Pakistan are available from a number of sources including Internet. These can make a terrain engine specifically useful for the military use.

9.2.1.3 Random Generated Height Maps

Using algorithms like Perlin Noise and Fractals mathematics, random terrain maps can be generated at run time and can provide the illusion of an Infinite terrain.

9.2.2 Weather Elements / Smoke

Weather elements like rain, snow hail storms can be generated using particle engines. Effects like smoke are necessary as well and can make a terrain engine more realistic. Clouds in this engine are 2D. In order to create infinite terrain, 3D clouds are a must. These can be generated from a number of ways; however all of these are resource heavy.

9.2.3 Results

The performance of this terrain engine can compete with many of the visual aspects of commercially available engines. With the inclusion of actual movements of sun and moon, and moon dates makes this one unique in this aspect, as most engines are of static nature. The creations of a terrain engine complete in all aspects and yet resource friendly is a major project of its own and can be made the scope of another degree project.

9.3 All Purpose Camera Analysis.

The Camera being one of the strongest features of this project fulfils almost all of the requirements ever needed for any graphics project. However the only point requiring improvement can be the encapsulation of its bare bones functionality in a more programmer friendly class structure. At present the camera code is too rough. With some effort it can be made more users friendly.

9.4 Gun Firing Simulation Analysis.

The firing simulation is mathematically correct, however further improvements can include the induction of effect of air resistance, wind speed and type/ shape of shell. However major effort at the mathematics end is required as very less literature is freely available. The present simulation is able to simulate the behaviour with minimum mathematical computation, however a complete research

project can be undertaken to simulate the behaviour under all situations. Further improvements can include behaviour of guided weapons like rockets, SAMs and long distance weapons.

9.5 Air Craft AI Analysis.

The aircraft follows the basic flight paths quiet well

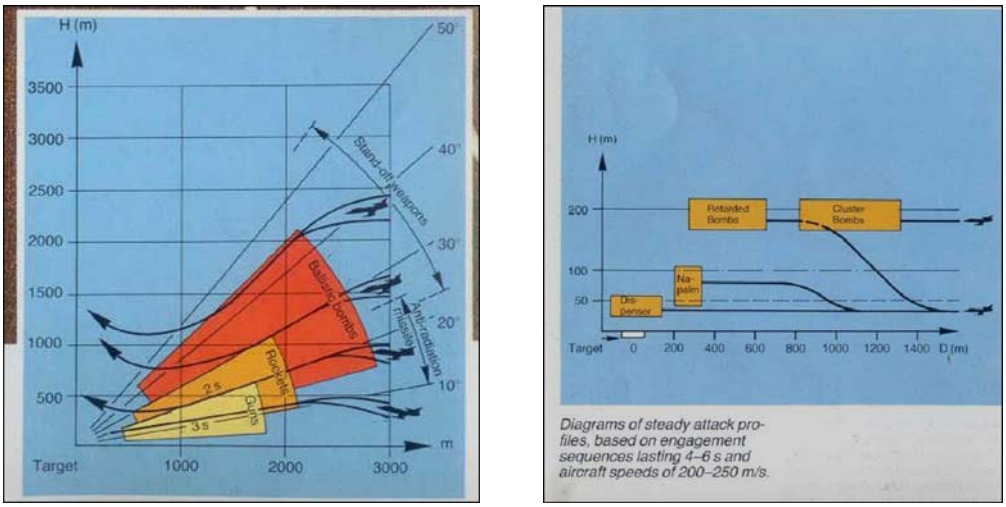

Fig 9.1 Basic Flight Profiles

A comparison can be seen using figure 9.1 and figure 9.2
Further improvements can the smoothing of flight path using the Bezier curves, which will make the flight even more realistic, however the technique is more computationally heavy.

9.6 Target Tracking and Results

The tracking is at present following all the rules of an Oerlikon weapon system. However the trainee is pitted against a non human enemy. Further improvements can the induction of a human controlled aircraft, on a distributed system. In a PC controlled environment multiple targets coming from different directions can be another valid improvement. Aircraft responding to air defence weapon firing is yet another area needing attention. However there is no end to

improvements and this project is a very humble start. A complete research project can be dedicated to simulating the behaviour of a realistic aircraft in bomber and fighter roles. Subsequently the training of troops will be more realistic.

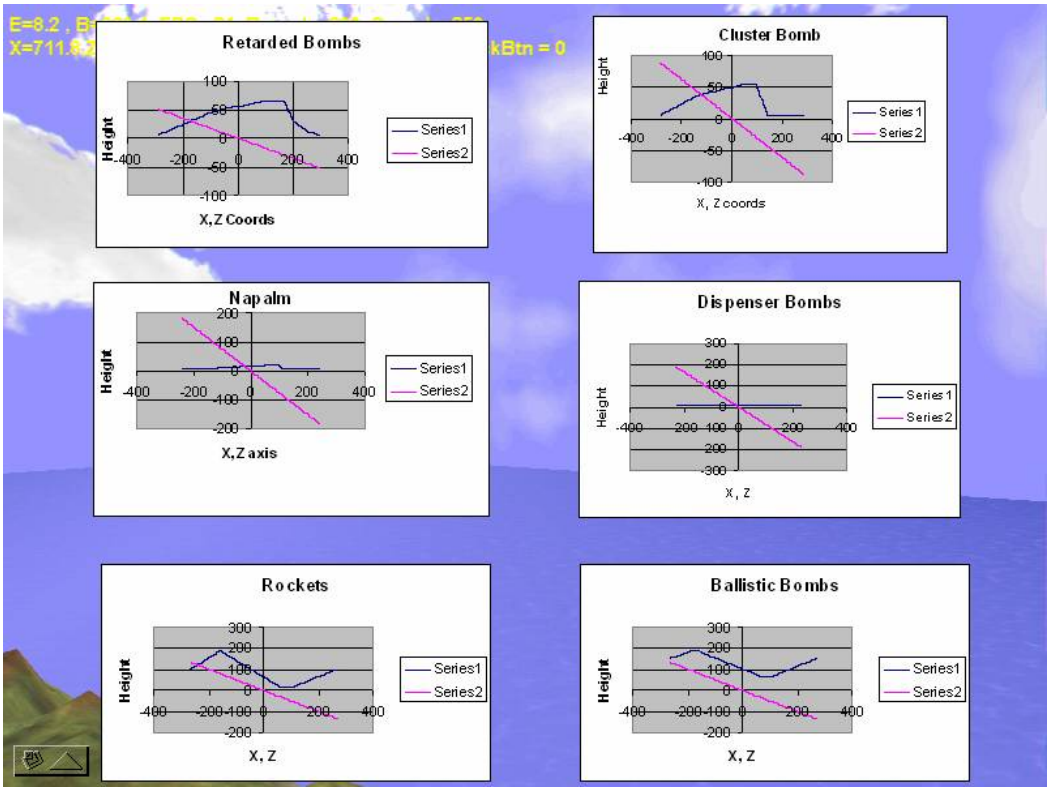

Fig 9.2 Correspondence to Basic Profiles

9.7 Resource Creation

The area of resource creation is a major area needing attention to the smallest details. The richest area of this project content wise, and most visible, was the most important as well. Any flaw in the visual content is immediately visible and thus cannot be compromised upon. The content creation of this project can be compared to any of the contents of a professional project, however small improvements can be made in the areas such as textures for all the 3D models, textures for the particle engine (if included) and induction of human models.

9.8 Gun Men System

The Gun Menu system is the core of this project, which make this project as the simulator of an Oerlikon based weapon system. It is complete in all respects; however following areas could not be integrated because of lack of available data or consensus from the interested organization as not being of immediate importance.

9.8.1 The Meteo Data.

The effect of Meteo data was not implemented as no technical data was available. It was decide that the effect will be incorporated on availability of technical data.

9.8.2 The Gun Procedures.

Many procedure exist which play an important role in the training of troops. However being a preliminary attempt and the first one of its type were not implemented in this version. Any further effort by another research group or a commercial organization can integrate these as well.

9.9 Conclusion

The Oerlikon simulator project was the first of its type and covered many areas, however room for improvement remains and many of the sub areas covered can be made full fledged degree projects of their own accord. If work is continued in this area, millions of dollars of national money can be saved. All modern armies are using simulations at tactical and strategic levels to train soldiers, juniors and senior commanders. Any equipment which is too costly or fragile to be used excessively in field needs a simulation. 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. The need is to embrace this new technology to its fullest, as soon as possible, as use it to its full

potential. Only then we stand a chance to compete with the modern armies of the World.

BIBLIOGRAPHY

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

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

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