VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi - 590 018.

MINI PROJECT REPORT

ON

"THE ATTACK"

Submitted in partial fulfilment of the requirements for the **Computer Graphics Laboratory (15CSL68)** ***course*** *of the 6th semester*

Bachelor of Engineering

In

Computer Science & Engineering

Under the guidance of

Ms. Sathyabhama R. **B.E., M.Tech.**
Asst.Professor, Dept. of CS&E.,
AIT, Chikkamagaluru.

Submitted By

HARSHA H K (4AI16CS027)

KARTHIK N L (4AI16CS031)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ADICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY
(Affiliated to VTU, Belagavi and Approved by AICTE, New Delhi)
Chikkamagaluru – 577102, Karnataka, India.

2018 – 2019

ADICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagavi and Approved by AICTE, New Delhi)

Chikkamagaluru- 577 102, Karnataka, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"THE ATTACK"** is a bonafide work carried out by **HARSHA HK (4AI16CS027), KARTHIK NL (4AI16CS031)** in partial fulfillment for the **Computer Graphics Laboratory (15CSL68)** course of 6th semester Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi during the academic year 2018-19. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of Project Work prescribed for the said Degree.

Signature of the Guide	**Signature of the HOD**
Ms.Sathyabhama R. B.E., M.Tech. Assistant Professor, Dept. of CS&E., AIT, Chikkamagaluru	**Dr. Pushpa Ravikumar** B.E., M.Tech., Ph.D., Professor & Head, Dept. of CS&E., AIT, Chikkamagaluru

External Examiner	**Signature with date**
1. _____	_____
2. _____	_____

ABSTRACT

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. **Computer graphics** is an art of drawing pictures, lines, charts, etc using computers with the help of programming.

In this project, we are aiming to simulate the situation where a convoy of busses carrying security people on the Jammu Srinagar National Highway was attacked by a vehicle-borne suicide bomber. We have created objects of **busses** carrying soldiers, **armed vehicle**, etc. We are making use of these objects for the simulation. The project proceeds by simulating the Air strike which was conducted by the Indian Air force as a response of the terrorist attack. This simulation consists of the objects such as **Fighter jet**, **terrorist base**, etc. This simulation is made user-interactive by using various keyboard keys such as arrow keys. We make us of key **C** to change the camera position. Key **S** is used to pause the simulation and key **M** is used to resume the simulation. And also use have created menus for better interaction.

ACKNOWLEDGEMENTS

We express our humble pranamas to his holiness a Jagadguru Padmabushana **Sri Sri Sri Dr. Balagangadharanatha MahaSwamiji** and Parama Poojya Jagadguru **Sri Sri Sri Dr. Nirmalanandanatha MahaSwamiji** who has showered his blessings on us for framing our career successfully.

We are deeply indebted to our honorable Principal **Dr. C. T. Jayadeva** for creating the right kind of care.

We express our deepest gratitude to **Dr. Pushpa Ravikumar**, Professor & Head, Department of Computer Science & Engineering, AIT, Chikkamagaluru for her valuable guidance, suggestions and constant encouragement without which success of our project work would have been difficult.

We are thankful to our guide **Ms. Sathyabhama R**, Assistant Professor, Department of Computer Science & Engineering, AIT, Chikkamagaluru for her inspiration and lively correspondence right from the beginning of our project work till its completion.

We would like to thank our beloved parents for their support, encouragement and blessings.

And last but not the least, we would be very pleased to express our heartfelt thanks to all teaching and non-teaching staff of CS&E department and our friends who have rendered their help, motivation and support.

HARSHA H K (4AI16CS027)

KARTHIK N L (4AI16CS031)

Table of Contents

List of figures

Chapter 1

Introduction

Pictorial synthesis of real/imaginary objects from computer-based models is Computer Graphics. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

1.1 History of Computer Graphics

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. This was started with the display of data on plotters and CRT. Computer Graphics is also defined as the study of techniques to improve the communication between user and machine, thus Computer Graphics is one of the most effective medium of communication between machine and user.

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeng. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

1.2 Introduction to Open GL

As a software interface for graphics hardware, OpenGL's main purpose is to render two and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. Primitives are defined by a group of one or more *vertices*. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normal, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way.

Figure 1.1 gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

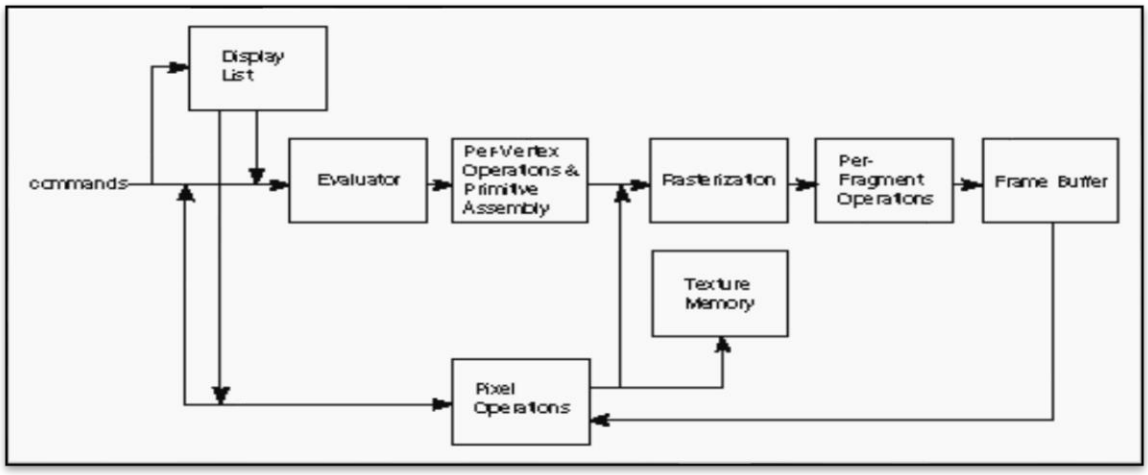

Fig 1.1: OpenGL Block Diagram

We can choose to accumulate some of commands in a display list for processing at a later time. The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment so produced is fed into the last stage, per-fragment operations, which perform the final operations on the data before it's stored as pixels in the frame buffer.

1.3 Objectives

Objectives of this project are

- To design THE ATTACK using C graphical functions.
- To provide user interaction using keyboard and mouse.

1.4 Organization of the report

Chapter 1 provides the information about the basics of OpenGL. In Chapter 2, all the OpenGL functions used in our program is described. Chapter 3 gives the idea of the project and its actual implementation. Chapter 4 discusses about the testing and limitations of the program. Chapter 5 concsludes by giving the direction for future enhancement.

1.5 Summary

The chapter discussed before is an overview about the computer graphics, its history and OpenGL interface. It even includes the OpenGL block diagram. The scope of study and objectives of the project are mentioned clearly. The organization of the report is been pictured to increase the readability. Further, coming up chapter depicts the OpenGL built-in functions used in project source code.

Chapter 2

System Requirement Specification

System requirement specification (SRS) presents the clear, brief also unambiguous explanation of the proposed methods. SRS allows the developers to check the system whether it satisfy the particular requirement or not. It stipulates the hardware and software prerequisite of the system. SRS assures that system is designed as mandatory at given time. It also describes the functional and non-functional requirements for the system.

2.1 Software Requirements

Operating system	: WINDOWS 10
Editor	: Microsoft Visual Studio 2015 or above
Programming language	: C++ with OpenGL

2.2 Hardware Requirements

Processor	: **Intel Core i5 Processor (8th Gen)**
Primary memory	: 4GB RAM or more
Frequency	: 2.40 MHz or more

Chapter 3

A Preview of OpenGL Functions

When we start drawing any graphics in OpenGL using C++ language we need a header file called <GL/glut.h>. The header file contains definitions and explanations of all functions and constants we'll need, whereas the graphics functions are kept in the library file. Both these files are provided as a part of TURBO CPP compliers.

3.1 Header Files

- **stdio.h:**

  This is a standard input header file which is used in any program. This file contains all the built-in functions like printf(), scanf(), fopen(), fclose() etc. It also contains data types and global variables. Some of the examples are BUFSIZ, EOF, and NULL etc.

- **stdlib.h:**

  This is also one the standard library header file which contains the entire standard library functions like exit, free, alloc, malloc etc. Some of the constants and data types are NULL, size_t, etc.

- **GL/glut.h:**

  This is very familiar library function of visual basic graphics library. This header file contains so many numbers of built in functions of a graphics library.

- **math.h:**

  This is also one the standard library header file which contains various mathematical functions and one macro. All the functions available in this library take double as an argument and return double as the result.

- **string.h:**

  This is the header in the C standard library for the C programming language which contains macro definitions, constants and declarations of functions and types used not only for string handling but also various memory handling functions.

3.2 OpenGL built-in functions

The different OpenGL functions used in our project are described as follows:

- **Name:** glBegin()

  Prototype: glBegin(glEnum mode);

  Description: Initiates a new primitive of type mode and starts collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON

- **Name:** glEnd()
  Prototype: void glEnd();

  Description: Terminates a list of vertices.

- **Name:** glutInit()
  Prototype: void glutInit ();

  Description: All Initializes the GLUT. The arguments from main are passed in and can be used by the application.

- **Name:** glutCreateWindow()

  Prototype: void glutCreateWindow();

  Description: Creates a window on the display. The string title can be used to label the window.

- **Name:** glutInitDisplayMode()

  Prototype: void glutInitDisplayMode();

  Description: Requests a display with the properties in mode. The mode is determined by the logical OR of options including the color model.

- **Name:** glutInitWindowSize()

  Prototype: void glutInitWindowSize();

  Description: Specifies the initial height and width of the window in pixels.

- **Name:** glutInitWindowPosition ()

  Prototype: void glutInitWindowPosition(int x, int y);

  Description: Specifies the initial position of the top-left corner of the window in pixels.

- **Name:** glutMainLoop()
  Prototype: void glutMainLoop();
  Description: Cause the program to enter an event-processing loop. It should be the last statement main.

- **Name**: glutDisplayFunc()
  Prototype: void glutDisplayFunc(void(*func)(void));
  Description: Registers the display functions func that is executed when the window must to be redrawn.

- **Name:** glutPostRedisplay()
  Prototype: void glutPostRedisplay();
  Description: Requests that the display callback be executed after the current callback returns.

- **Name:** glMatrixMode()
  Prototype: void glMatrixMode(GLenum mode);
  Description: Specify matrix will be affected by subsequent transformation mode can be GL_MODELVIEW, GL_PROJECTION.

- **Name**: glutKeyboardFunc()
  Prototype: void glutKeyboardFunc(void(*func)(void));
  Description: Sets the keyboard interaction with the current window.

- **Name:** glFlush()
  Prototype: glFlush();
  Description: The call to glFlush ensures that points are rendered to the screen.

- **Name:** glClear()
  Prototype: glClear(GL_COLOR_ BUFFER_BIT|GL_DEPTH_BUFFER_BIT**);**
  Description: To refresh the color buffer and depth buffer so that if algorithm stores information in the depth buffer, we must clear this buffer whenever we wish to redraw the display.

- **Name:** glutSwapBuffers()
  Prototype: glutSwapBuffers();
  Description: Swaps the buffers of the current window if double buffered Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers() promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers() is called.

- **Name:** glutInitWindowSize()
  Prototype: glutInitWindowSize();
  Description: It is used to create the current, initial window position and size.

- **Name:** glutMotionFunc()
  Prototype: void glutMotionFunc(void (*f)(void));
  Description: This function reports the mouse position when the mouse is dragged starting from within your window.

- **Name:** glutSpecialFunc()
  Prototype: void glutSpecialFunc(void (*f)(void));
  Description: Registers a callback for OpenGLUT to call when the user presses "special" keys on the keyboard. The special callback handles some additional keys that are not covered under plain "keyboard" events. The key that is passed to the callback is one of an enumerated set.

- **Name:** glutSolidSphere()
  Prototype: void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
  Description: The glutSolidSphere() function draws a shaded sphere centered at the origin. The surface is created from quadrangles (except for triangles as degenerate quads at the poles) in a longitude/latitude pattern. The equatorial great circle lies in the xy-plane and is centered on the origin.

- **Name:** glutIdleFunc()

  Prototype: void glutIdleFunc(void (*f)(void));

  Description: It is used to register the display callback function 'f' that is executed whenever there are no other events to be handled.

- **Name:** glutSolidCube()

  Prototype: void glutSolidCube(GLdouble width);

  Description: The glutSolidCube() function draws a solid-shaded cube with side-length given by width. The vertices of the cube are at (+/- width/2, +/- width/2, +/- width/2), so that the cube is centered at the origin.

- **Name:** glutSolidTorus ()

  Prototype: void glutSolidTorus(GLdouble dInnerRadius, GLdouble dOuterRadius, GLint nSides, GLint nRings);

  Description: This function effectively wraps a cylinder with nSides slats and bends it at nRings facets around a circular path, forming a torus, or "donut". The center is at the origin and the "path" rings around the z axis.

- **Name:** gluCylinder()

  Prototype: void gluCylinder(GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks);

  Description: gluCylinder draws a cylinder oriented along the z axis. The base of the cylinder is placed at z = 0. Like a sphere, a cylinder is subdivided around the z axis into slices, and along the z axis into stacks.

- **Name:** gluDisk()

  Prototype: void gluDisk(GLUquadric* quad, GLdouble inner, GLdouble outer, GLint slices, GLint loops);

  Description: gluDisk renders a disk on the z = 0 plane. The disk has a radius of outer, and contains a concentric circular hole with a radius of inner. If inner is 0, then no hole is generated. The disk is subdivided around the z axis into slices (like pizza slices), and also about the z axis into rings (as specified by slices and loops, respectively).

- **Name:** gluLookAt()

  Prototype: void gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY,GLdouble upZ);

  Description: gluLookAt creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an *UP* vector.The matrix maps the reference point to the negative *z* axis and the eye point to the origin. When a typical projection matrix is used, the center of the scene therefore maps to the center of the viewport. Similarly, the direction described by the *UP* vector projected onto the viewing plane is mapped to the positive *y* axis so that it points upward in the viewport. The *UP* vector must not be parallel to the line of sight from the eye point to the reference point.

- **Name:** main()

  Prototype: int main(int argc, char** argv);

  Description: A special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.

Chapter 4

Design and Implementation

4.1 Introduction

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development.

This Project is implemented using OpenGl, which is proven to be a very efficient tool in the field of computer graphics, programming is done under Windows 10 platform. glut.h library is used to create the objects and to translate them. C++ programming language is used to implement the entire code. Interface to the program is provided with the help of input device keyboard and mouse.

4.2 Overall Design Process

There are few graphical functions used in our project. The arrow keys are used to select the required square. The Enter key is used to place the bomb on the selected square.

4.3 User defined functions:

- **Name:** LoadBMP()
  Prototype: GLuint LoadBMP(const char* fileName);
  Description: To load the bitmap file into a texture variable.

- **Name:** Draw_Skybox()
  Prototype: void Draw_Skybox(float x, float y, float z, float width, float height, float length);
  Description: To load sky bitmaps into the surrounding quads.

- **Name:** draw_ground ()
  Prototype: void draw_ground();
  Description: To create ground quad and load grass bitmap into it.

- **Name:** drawperson()
  Prototype: void drawperson(int i);
  Description: To draw 3D object of a person.

- **Name:** drawbus()
  Prototype: void drawbus(GLfloat polytype);
  Description: To draw 3D object of a bus.

- **Name:** drawtree()
  Prototype: void drawtree();
  Description: To draw 3D object of a tree.

- **Name:** draw_road()
  Prototype: void draw_road();
  Description: To draw 2D plane of a road.

- **Name:** drawbomb()
  Prototype: void drawbomb(float ychng,float zchng);
  Description: To draw 3D object of bombs which are carried by SUV.

- **Name:** drawflame()
  Prototype: void drawflame();
  Description: To show fire effect when the bombs explode.

- **Name:** drawbckgrnd()
  Prototype: void drawbckgrnd();
  Description: To show introduction and description bitmaps.

- **Name:** displaystrike()
  Prototype: void displaystrike();
  Description: To draw 3D object of a plane and to simulate the air-strike on terrorist base.

- **Name:** drawbase()

  Prototype: void drawbase();

  Description: To draw 3D object of terrorist base.

- **Name:** display()
  Prototype: void display();

  Description To render everything about the attack onto the screen.

- **Name:** displayReshape()
  Prototype: void displayReshape();

  Description: To handle a resize event.

- **Name:** windowSpecial()
  Prototype: void windowSpecial(int key, int x, int y);

  Description: To handle special key hit events.

- **Name:** kb()

  Prototype: void kb(unsigned char key, int x, int y);

  Description: To handle normal key hit events.

- **Name:** handleMouse()

  Prototype: void handleMouse(int x, int y);

  Description: To handle mouse events.

- **Name:** idle()

  Prototype: void idle();

  Description: This method will be executed everytime the mainloop loops.

- **Name:** Menu()

  Prototype: void Menu(int action);

  Description: To register menu entries to mouse right button.

4.4 Algorithm

Algorithm Attack

Step 1: BEGIN

Step 2: Display Introduction.

Step 3: Call SUV_Attack Algorithm.

Step 4: Call Air_Strike Algorithm.

Step 5: END

Algorithm SUV_Attack

Step 1: BEGIN

Step 2: Start Simulation.

Step 3: While not crash then

Step 4: If ‘s’ is pressed then

Step 5: pause movement

Step 6: If ‘m’ is pressed then

Step 7: resume movement

Step 8: If ‘c’ is pressed then

Step 9: change camera view

Step 10: If arrow keys are pressed then

Step 11: navigate accordingly

Step 12: end while.

Step 13: Display Flames

Step 14: END

Algorithm Air_Strike

Step 1: BEGIN

Step 2: Start Simulation.

Step 3: While 'd' not pressed do

Step 4: If 'c' is pressed then

Step 5: set camera to plane inner view

Step 6: If 'b' is pressed then

Step 7: Show button target view

Step 8: If 't' is pressed then

Step 9: Show top view.

Step 10: If arrow keys are pressed then

Step 11: navigate accordingly

Step 12: else

Step 13: set camera to plane rear view

Step 14: end While

Step 15: Drop missiles

Step 16: If missiles hit the target

Step 15: return

Step 16: else

Step 17: goto Step 2

Step 18: END

4.5 Flowcharts

- Figure 4.1 shows the flowchart for the whole project which consists of 2 sub routines SUV_Attack and Air_Strike. Figure 4.2 shows the flowchart for SUV_Attack algorithm. This proceeds by starting the simulation and waiting until the vehicle crashes with the buses. Figure 4.3 shows the flowchart implemented for the Air_Strike algorithm. This flowchart proceeds by starting the jet simulation and continues by displaying the plane until inner camera is switched on. Later based on the arrow keys, the plane will be navigated accordingly.

Flowchart of The Attack

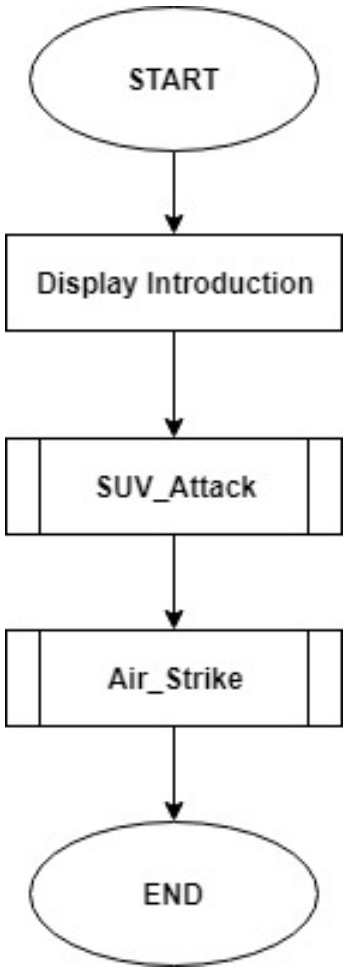

Fig 4.1: Flowchart of The Attack

Flowchart of SUV_Attack

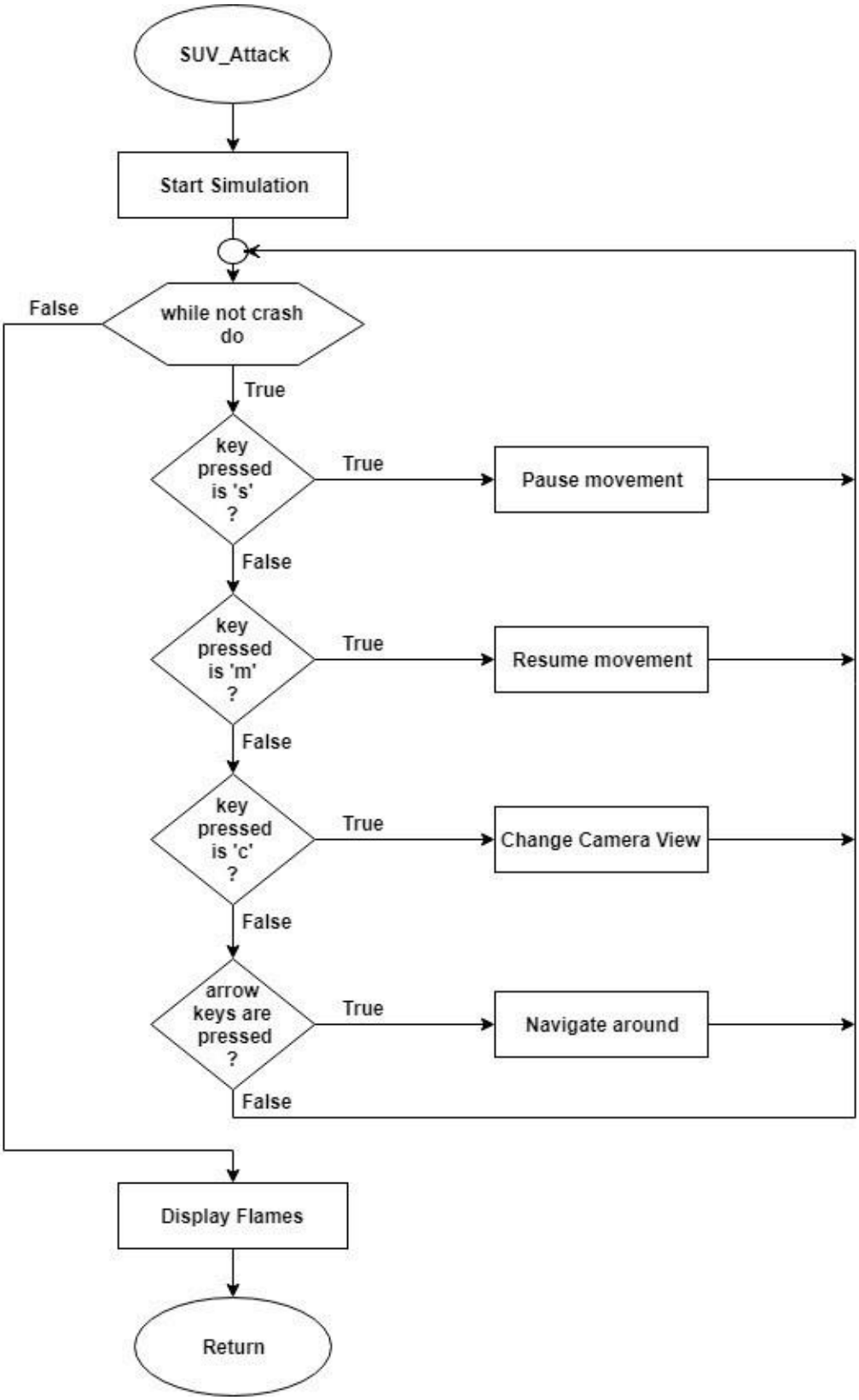

Fig 4.2: Flowchart of SUV_Attack

Flowchart of Air_Strike

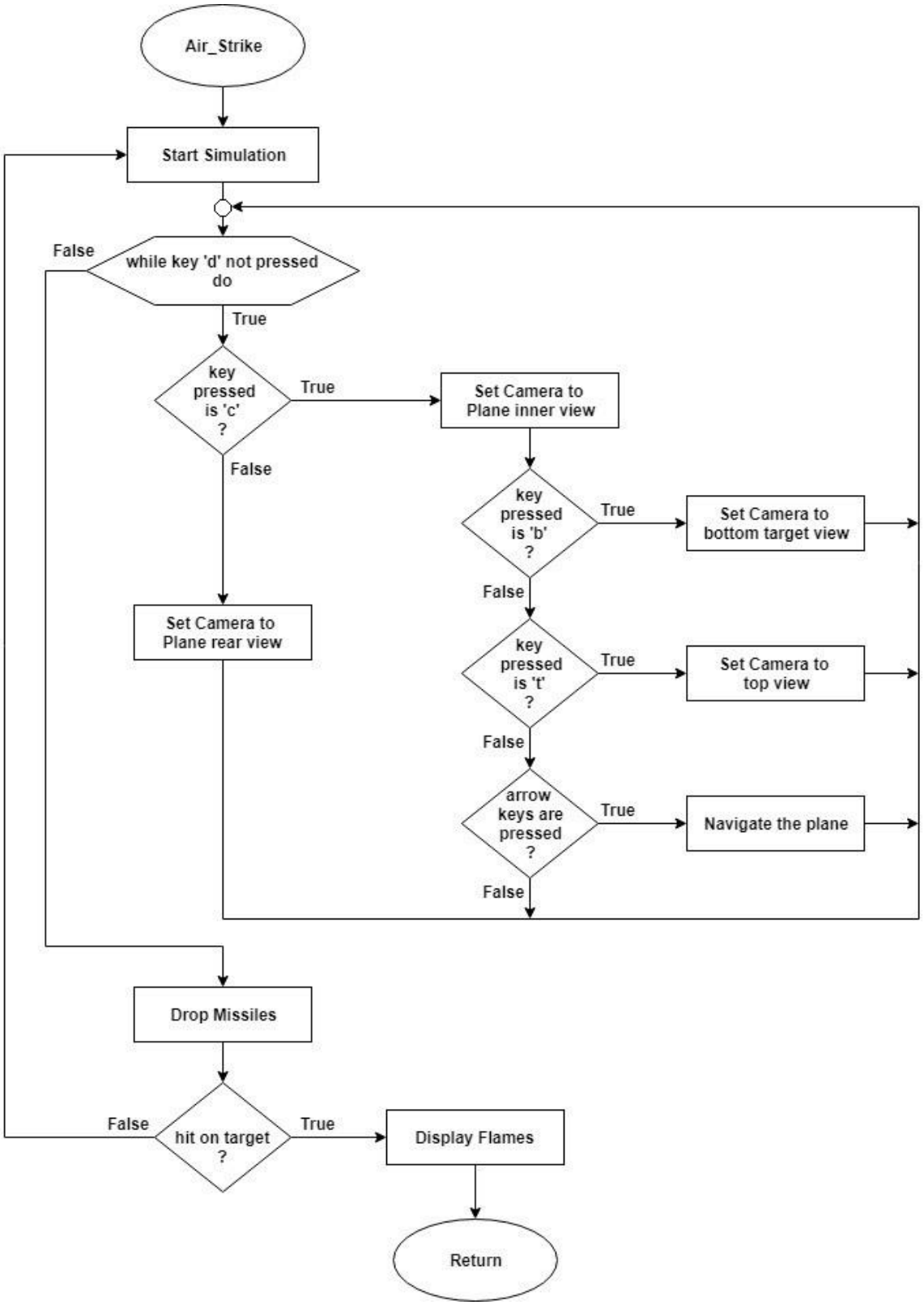

Fig 4.3: Flowchart of Air_Strike

4.6 Logic code

- **Logic code to load bitmap files**

```
GLuint LoadBMP(const char* fileName)
{
                FILE *file;
                unsigned char header[54], *data;
                unsigned int dataPos, size, width, height;
                fopen_s(&file, fileName, "rb");
                fread(&header, sizeof(char), 54, file);
                dataPos = *(int*)&(header[0x0A]);
                size = *(int*)&(header[0x22]);
                width = *(int*)&(header[0x12]);
                height = *(int*)&(header[0x16]);
                if (size == NULL)
                       size = width * height * 3;
                if (dataPos == NULL)
                       dataPos = 54;
                data=new unsigned char[size];
                fread(data, sizeof(char), size, file);
                fclose(file);
                GLuint texture;
                glGenTextures(1, &texture);
                glBindTexture(GL_TEXTURE_2D, texture);
                glTexParameterf(GL_TEXTURE_2D,
               GL_TEXTURE_MAG_FILTER, GL_NEAREST);
                glTexParameterf(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER, GL_NEAREST);
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height,
                0, GL_BGR_EXT, GL_UNSIGNED_BYTE, data);
                return texture;
}
```

- **Logic code to create ground quad**

```
void draw_ground()
{
                glEnable(GL_TEXTURE_2D);
                glBindTexture(GL_TEXTURE_2D, grass);
                glColor3f(0, .8, 0);
                glBegin(GL_QUADS);
                glTexCoord2f(0.0f, 0.0f);
                glVertex3f(2000, -2,  18000);
                glTexCoord2f(50.0f, 0.0f);
                glVertex3f(2000,-2, -18000);
                glTexCoord2f(50.0f,50.0f);
                glVertex3f(-18000,-2,18000);
                glTexCoord2f(0.0f, 50.0f);
                glVertex3f(-18000, -2,18000);
                glEnd();
                glDisable(GL_TEXTURE_2D);
}
```

- **Logic code of main() function**

```
int main(int argc, char** argv)
{
                glutInit(&argc, argv);
                glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);
                glutInitWindowPosition(0, 0);
                glutInitWindowSize(1500, 1000);
                glutCreateWindow("The ATTACK");
                initLights();
                initLightsforGW();
                initSky();
                glutReshapeFunc(displayReshape);
                glutDisplayFunc(display);
                glutKeyboardFunc(kb);
                glutMotionFunc(handleMouse);
                glutPassiveMotionFunc(passiveMouse);
                glutIdleFunc(idle);
                addmenu();
                glutSpecialFunc(windowSpecial);
                glutMainLoop();
                return 0;
}
```

Chapter 5

Results and Discussion

The project is compiled and executed on Visual Studio 2017. We have attached few screen shots in here to show the working of our project.

5.1 Snapshots of project THE ATTACK

Snapshot-1:

Figure 5.1, shows the introduction page when the project is executed.

Fig 5.1: Introduction page.

Snapshot-2:

Figure 5.2, shows a convoy of CRPF buses.

Fig 5.2: Convoy of CRPF buses.

Snapshot-3:

Figure 5.3, shows the vehicle used for the attack.

Fig 5.3: Heavily armed SUV.

Snapshot-4:

Figure 5.4, shows view of the blast.

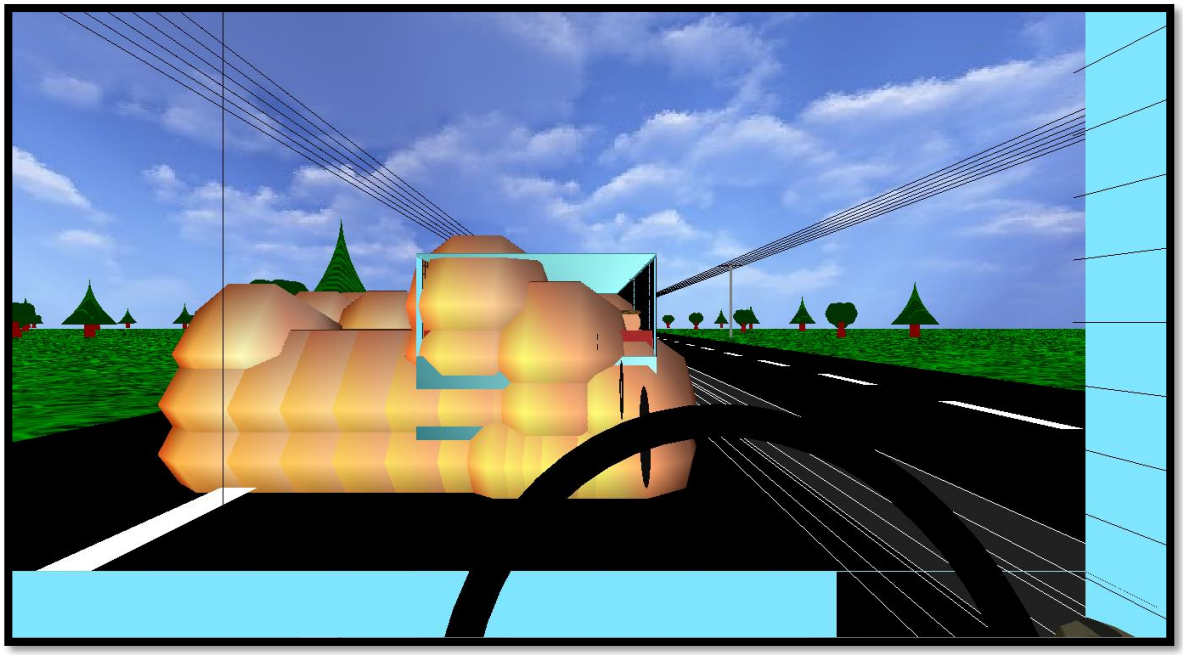

Fig 5.4: Explosion of bombs.

Snapshot-5:

Figure 5.5, shows the help page when 'h' is pressed.

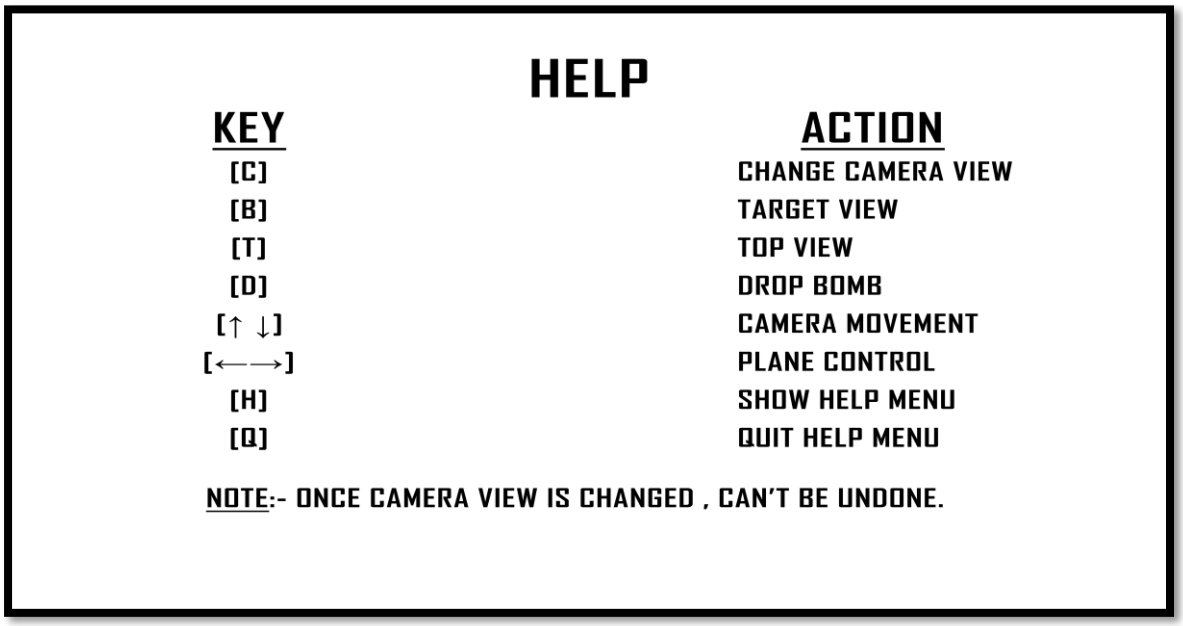

Fig 5.5: Instruction page

Snapshot-6:

Figure 4.6, shows the fighter jet ready to take-off.

Fig 5.6: Fighter jet.

Snapshot-7:

Figure 4.7, shows the view of terrorist base from the jet.

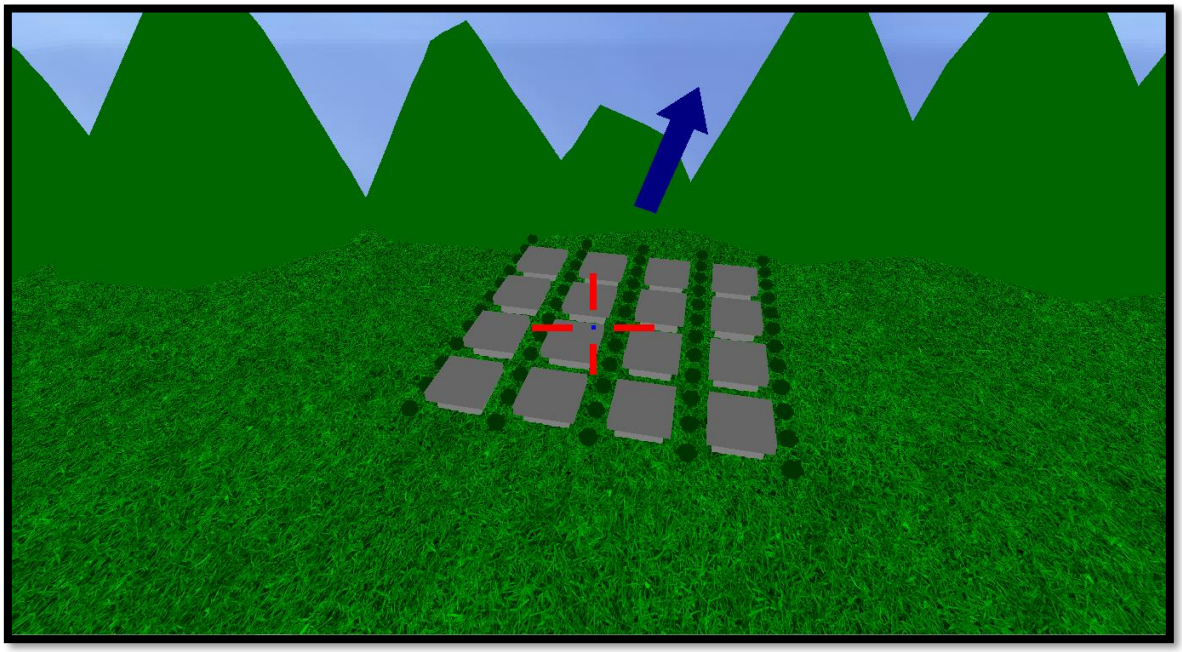

Fig 5.7: Terrorist base.

Snapshot-8:

Figure 5.8, shows the view of fighter jet dropping missiles on terrorist base.

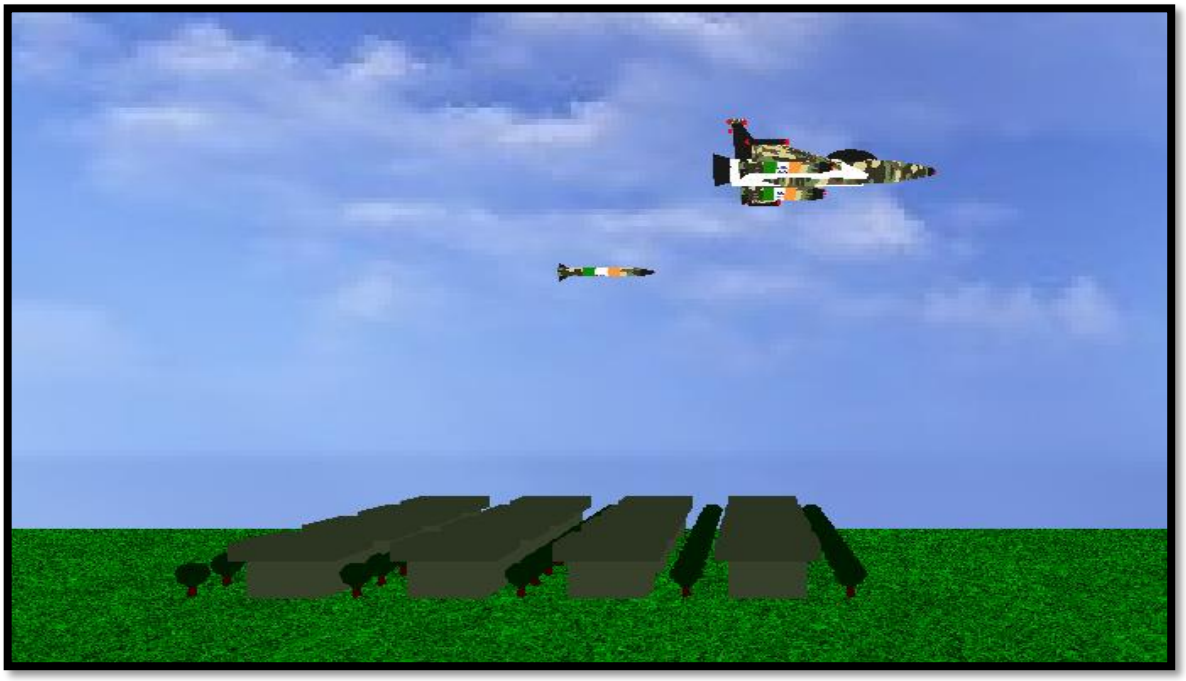

Fig 5.8: Dropping missiles.

Snapshot-9:

Figure 5.9, shows the scene of terrorist base being destroyed.

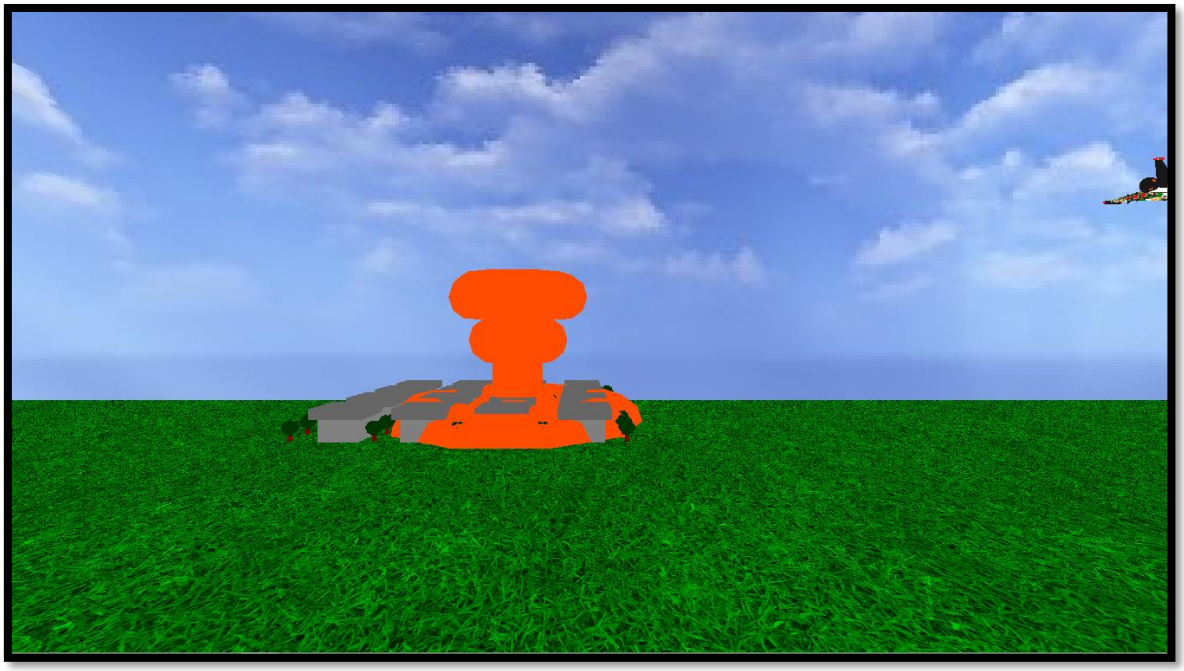

Fig 5.9: Explosion.

Chapter 5

Conclusion and

Conclusion

This project is all about the illustrations of OpenGL and GLUT. We have used suitable OpenGL functions to develop objects. Aim of this project is to visualize the terrorist attack that occurred at Pulwama and also the revenge through Air-strike by the Indian Airforce. All the functions of the project are the outcome of the openGL built-in in function calls.

References

[1] Edward Angel, “Interactive Computer Graphics”, Pearson education, Fifth Edition.

[2] “Computer Graphics Using OpenGL” - F.S.Hill, Jr. 2nd Edition, Pearson Education, 2001

[3] www.opengl.org/documentation/specs/gut/spec3/spec3.html

[4] www.OpenGL.org/recources/code/samples

[5] https://linux.die.net/

[6] https://www.khronos.org/

[7] https://www.learnopengl.com/

Appendix A

OpenGL Routines:

- **void glBegin (glEnum mode) :**
  Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES, and POLYGON

- **void glEnd () :**
  Terminate a list of vertices.
- **void glutInitWindowPositin(intx,int y) :**
  Specify the initial position of the top-left corner of the window in pixel.
- **void glutMainLoop() :**
  Cause the program to enter an event-processing loop. It should be the last statement in Main
- **void glutDisplayFunc(void (*func)(void)) :**
  Registers the display functions func that is executed when the window needs to be redrawn.
- **void glutSwapBuffers() :**
  Swaps the front and back buffers.
- **void glutAddmenuEntry (char *name, int value) :**
  Add the entry with the string name displayed to the current menu. Values are returned to the menu callback when the entry is selected.
- **void glMatrixMode (GLenum mode) :**
  Specifies matrix will be affected by subsequent transformations. Mode can be GL_MODELVIEW,GL_PROJECTION

- **void LoadIdentity() :-**

  Set the current transformation matrix to an identity matrix.