VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI - 590018

Mini Project Report

on

WAR SCENARIO

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

KARTHIK K.K

1BG19CS043

Guided by

Dr. Kavitha Jayaram

Vidyayāmruthamashnuthe

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.

All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021

Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www.bnmit.org

Department of Computer Science and Engineering

2021-22

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

Department of Computer Science and Engineering

Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the Mini Project entitled **WAR SCENARIO** carried out by

KARTHIK K.K (1BG19CS043) a bonafide student of VI Semester B.E., **B.N.M Institute of Technology** in partial fulfillment for the Bachelor of Engineering in COMPUTER SCIENCE AND ENGINEERING of the **Visvesvaraya Technological University**, Belagavi during the year 2021-22. It is certified that all corrections/ suggestions indicated for internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of Computer Graphics Laboratory with Mini Project as prescribed for the said degree.

Dr. Kavitha Jayaram	**Dr. Chayadevi M L**
Associate Professor	**Professor & HOD**
Department of CSE	**Department of CSE**
BNMIT, Bengaluru	**BNMIT, Bengaluru**

	Name	**Signature**
Examiner1:		
Examiner2:		

ABSTRACT

Today, computer graphics is a core technology in video games, digital photography, film, cell phone and computer displays, and many specialized applications. The main objective of this project is to display the working of maze game using C Language with OpenGL library function. The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.

In addition to being language-independent, OpenGL is also cross-platform. The specification says nothing on the subject of obtaining and managing an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

This project is based on War Scenario . This project basically has two tanks which are moving. The are made to move using the Translate function.The tanks have a rotating gun.The rotation has been made possible because of the use of rotation function in opengl.

The menu function has been implemented here . Hence we have 3 menus which will be display on right click of the mouse

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project.

I would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMEI, Bengaluru for providing the excellent environment and infrastructure in the college.

I would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to sincerely thank **Dr. S Y Kulkarni**, Additional Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to express my gratitude to **Prof. Eishwar N Maanay**, Dean, BNMIT, Bengaluru for his relentless support and encouragement.

I would like to thank **Dr. Krishnamurthy G N**, Principal, BNMIT, Bengaluru for his constant encouragement.

I would like to thank, **Dr. Chayadevi M L**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

I would also like to thank **Dr. Kavitha Jayaram,** Associate Professor, Department of Computer Science and Engineering for providing me with her valuable insight and guidance wherever required throughout the course of the project and its successful completion.

KARTHIK K.K

1BG19CS043

TABLE OF CONTENTS

List of Figures

Chapter 1 INTRODUCTION

1.1 Overview

Interactive computer graphics provides us with the most natural means of communicating information through a computer. Over the years advancements in computer graphics have enabled us not only to make pictures of real-world objects but also visualize abstract, synthetic objects such as mathematical surfaces and of data that have no inherent geometry, such as survey results. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

1.2 Problem Statement

To develop a war scenario which shows movement of war tanks with rotating guns and planes moving.

1.3 Motivation

War Scenario help to get a 3D view of the war field . The soldier can plan his strageries in the battle field based on this simulation.The ability to develop this visualization using C programming and the OpenGL API serves as a motivation to develop this application.

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992; applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which may be called by the client program, alongside a set of named integer constants. In addition to being language-independent, OpenGL is also cross-platform.

Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of several game-development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL UtilityToolkit (GLUT), later superseded by freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

Architecture of OpenGL API has been described in figure 1.1.

- **Display Lists**:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

- **Evaluators:**

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

- **Per Vertex Operations:**

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

- **Primitive Assembly:**

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

- **Pixel Operation:**

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

- **Rasterization:**

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Colour and depth values are assigned for each fragment square.

- **Fragment Operations**:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

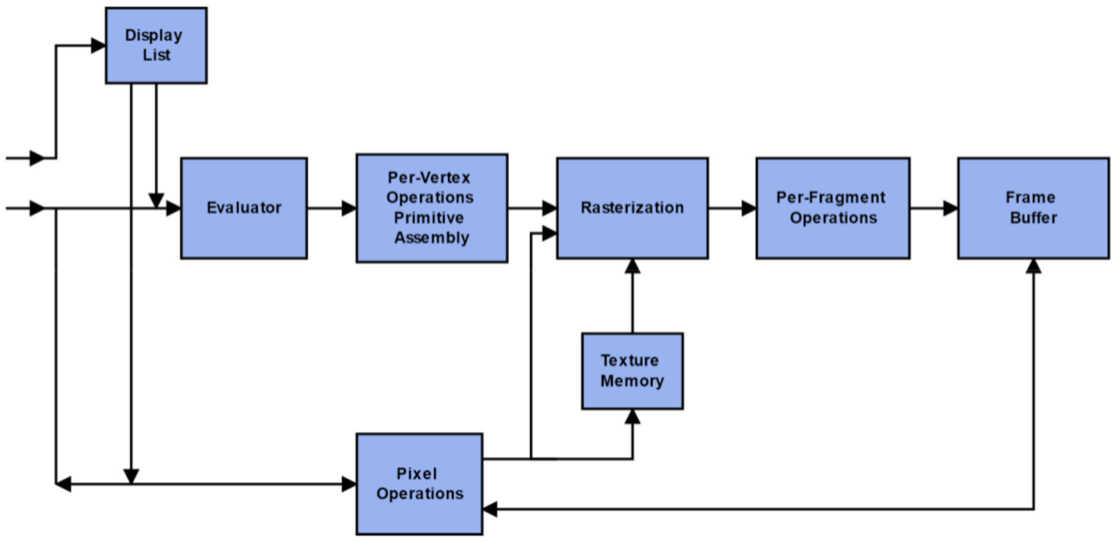

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain.

We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating colour, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term ―computer graphics‖ was coined in 1960 by William Fetter, a designer at Boeing, to describe his own job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland‘s Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving and constraining – with use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad‘s innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much of GUI on its thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms and the representation of points by homogeneous coordinates, the latter crucially presagingthe foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

The seventies brought the Z-buffer for hidden surface removal, texture mapping, Phong‘s lighting model – all crucial components of the OpenGL API (Application Programming Interface) we‘ll be using soon – as well as keyframe-based animation. Photorealistic rendering of animated movie keyframes almost invariably deploys ray tracers, which were born in the seventies too.

Through the nineties, as well, the use of 3D effects in movies became pervasive. The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer-generated – no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was cinema without film. Quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross language 3D graphics API, by Silicon Graphics. OpenGL is actually a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer‘s Hierarchical Interactive Graphics System) first brought 3D graphics programming to the ―masses‖. What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It‘s only among game developers that Microsoft‘s proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller and cheaper, continually pushing erstwhile high-end software down market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

- **Computer Aided Design (CAD):**

Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design it is known as mechanical design.

automation (MDA) or computer-aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

- **Computer Simulation**

Computer simulation is the reproduction of the behavior of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models, computer simulations have become a useful tool for the mathematical modeling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

- **Digital Art**

Digital art is an artistic work or practice that uses digital technology as part of the creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture and music/sound art, while new forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

- **Virtual Reality**

Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. A person using virtual reality equipment is able to look around the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

- **Video Games**

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, virtual reality headset or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades, but declined in popularity as other, more affordable platforms became available. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktop or handheld computing devices.

Chapter 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application :

- Operating System: Windows 10
- Compiler: GNU C/C++ Compiler
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Corerecommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter 3
SYSTEM DESIGN

4.1 Proposed System

This project is based on War Scenario . This project basically has two tanks which are moving. The are made to move using the Translate function.The tanks have a rotating gun.The rotation has been made possible because of the use of rotation function in opengl.
The menu function has been implemented here . Hence we have 3 menus which will be display on right click of the mouse

To achieve three dimensional effects, OpenGL software is proposed. It is software which provides a graphical interface. It is an interface between application program and graphics hardware. The advantages are:

- OpenGL is designed as a streamlined.
- It is a hardware independent interface, it can be implemented on manydifferent hardware platforms.
- With OpenGL, we can draw a small set of geometric primitives such aspoints, lines and polygons etc.
- It provides double buffering which is vital in providing transformations.
- It is event driven software.
- It provides call back function.

4.2 Data Flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data- flow diagram has no control flow, there are no decision rules and no loops. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes. The data-flow diagram is part of the structured-analysis modelling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan.

Fig 4.1 shows the data flow diagram of the opengl program written in C language

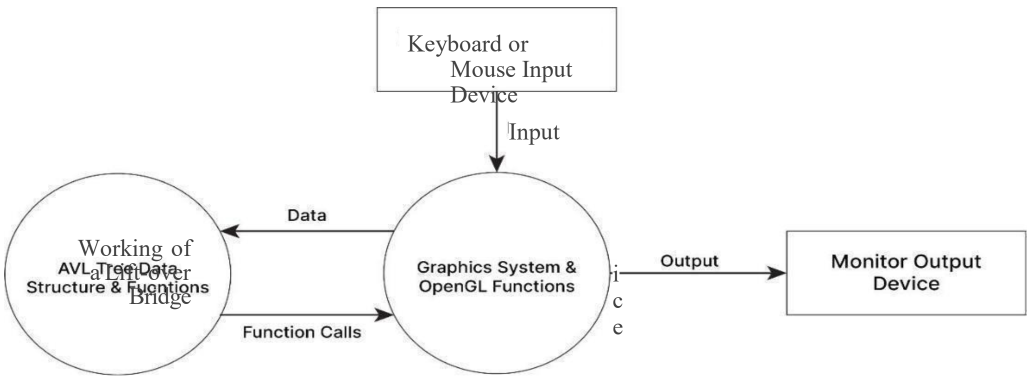
Keyboard or Mouse Input Device
Input
Data
Working of a Lift-over Bridge
AVL Tree Data Structure & Functions
Function Calls
Graphics System & OpenGL Functions
ice
Output
Monitor Output Device

Figure 4.1 Level 0 Dataflow Diagram of the Proposed System

Figure 4.1 shows the Level 0 Dataflow diagram of the proposed application for a Lift-over bridge. The keyboard and mouse devices are used for input to the application.

The graphics system processes these user keyboard/mouse interactions using built in OpenGL functions like glutKeyboardFunc(void *) and glutMouseFunc(*void). Lift-over Bridge is constructed in the memory using the user inputs sent to it by the Graphics System.

4.2 Flowchart

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows.

Fig 4.2 shows the data flowchart of the opengl program written in C language

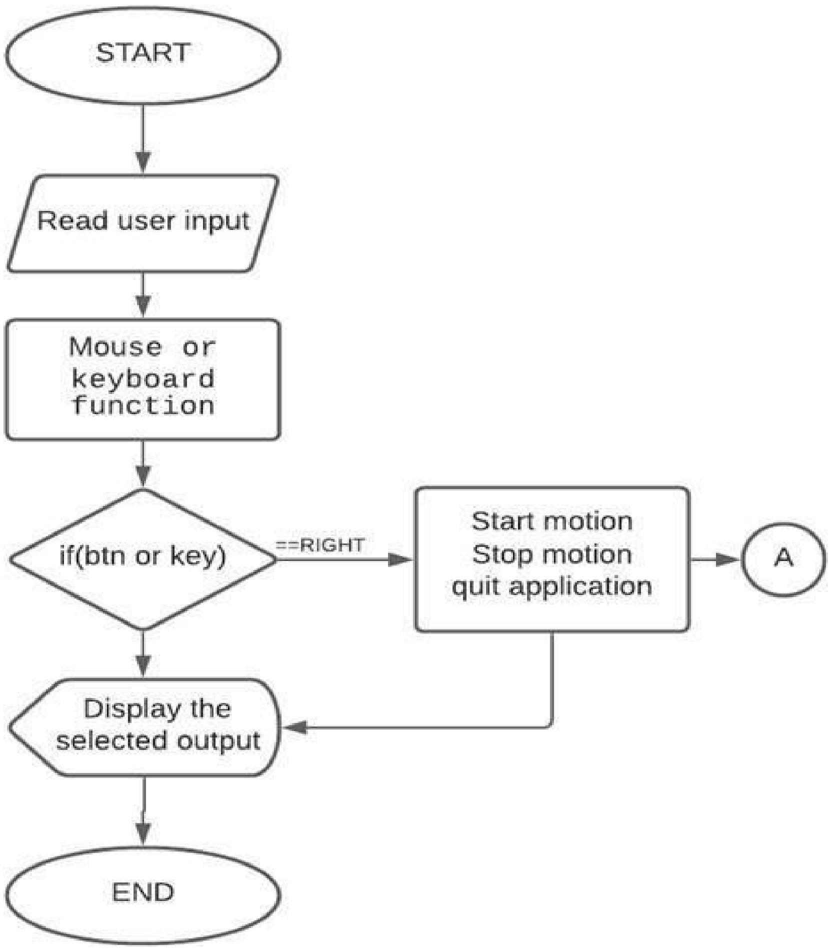

Figure 4.2 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **Void plane():**

  Displays the fighter jet on the screen

- **Void tank1():**

  Displays the tank 1 on the screen

- **Void tank1():**

  Displays the tank 2 on the screen

- **Void tankgun1():**

  Displays the gun attached to the tank1

- **Void tankgun1():**

  Displays the gun attached to the tank2

- **Void Buildings():**

  Displays the buildings .

- **Void display():**

  This function basically displays all the above described functions on the screen and translate and rotate the tankguns as we flush the output onto the screen form the frame buffer.

- **Void myinit():**

  This function is to specify the matrix mode and background color.

- **Void menu():**

Contains Menu to Move the Tank,

- **Void main():**

This function puts the whole program together. It says which function to execute first and which one at the end. However, here we have used int main () since eclipse expects the main to have a return value.

5.2 High Level Code

5.2.1 Built-In Functions

- **Void glColor3f (float red, float green, float blue):**

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. The _f' gives the data type float. The arguments are in the order RGB (Red, Green and Blue). The color of the pixel can be specified as the combination of these 3 colors.

- **Void glClearColor(int red, int green, int blue, int alpha):**

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

- **Void glMatrixMode(GLenum mode):**

Where –mode| specifies which matrix stack is the target for subsequent operations.

- **Void glutKeyboardFunc():**

Where func () is the new keyboard callback function. glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. Prototype is as given below:

Void glutKeyboardFunc (void (*func) (unsigned char key, int x, int y));

- **Void GLflush():**

GLflush () empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finitetime.

- **Void viewport(GLint x, GLint y, GLsizei width, GLsizei height):**

Here, (x, y) specifies the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0).Width, height: Specifies the width and height of the viewport. When a GL context is first attached to a surface (e.g. window), width and height are set to the dimensions of that surface.

- **glutPostRedisplay()**

GlutPostRedisplay, glutPostWindowRedisplay — marks the current or specified window as needing to be redisplayed.

- **glTranslatef ()**

Translation is done by adding the required amount of translation quantities to each of points of the objects in the selected area. If P(x,y) be the a point and (tx, ty) translation quantities the nthe translated point is given by glTranslatef(dx,dy,dz) ;

- **glRotatef ()**

The rotation of an object by an angle 'a' is accomplished by rotating each of the pointsof the object. The rotated points can be obtained using the OpenGL function glRotatef(angle, vx,vy,vz);

- **void glutInit (int argc, char**argv):**

GlutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

- **Void glutReshapeFunc (void (*func) (int width, int height)):**

GlutReshapeFunc sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established.

- **void glutMainLoop(void);**

GlutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never stop.

5.2.2 User Implementation

Function for start screen

```
  #include<GL/glut.h>
  #include<stdio.h>
  int x, y;
  int rFlag = 0;
 GLfloat angle = 45.0f;
 int y3 = 0;
void buildings()
{
        glColor3f(1.0, 1.0, 0.6);
        glBegin(GL_QUADS);
        glVertex2f(-479, -100);
        glVertex2f(-479, 70);
        glVertex2f(-420, 70);
        glVertex2f(-420, -100);
        glEnd();

        glBegin(GL_QUADS);
        glColor3f(1.0, 0.5, 0.6);
        glVertex2f(-420, -100);
        glVertex2f(-420, 80);
        glVertex2f(-380, 80);
        glVertex2f(-380, -100);
        glEnd();

        glBegin(GL_QUADS);
        glColor3f(1.0, 0.1, 0.6);
        glVertex2f(-380, -100);
        glVertex2f(-380, 50);
        glVertex2f(-340, 50);
        glVertex2f(-340, -100);
        glEnd();

        glColor3f(1.0, 1.0, 0.6);
        glBegin(GL_QUADS);
        glVertex2f(-179, -100);
        glVertex2f(-179, 70);
        glVertex2f(-139, 70);
        glVertex2f(-139, -100);
        glEnd();

        glBegin(GL_QUADS);
        glColor3f(1.0, 0.5, 0.6);
        glVertex2f(-139, -100);
        glVertex2f(-139, 80);
        glVertex2f(-99, 80);
 glVertex2f(-99, -100);
 glEnd();

 glBegin(GL_QUADS);
 glColor3f(1.0, 0.1, 0.6);
 glVertex2f(-99, -100);
 glVertex2f(-99, 40);
 glVertex2f(-59, 40);
 glVertex2f(-59, -100);
 glEnd();
 glColor3f(1.0, 1.0, 0.6);
 glBegin(GL_QUADS);
 glVertex2f(-179, -100);
 glVertex2f(-179, 70);
 glVertex2f(-139, 70);
 glVertex2f(-139, -100);
 glEnd();
 glBegin(GL_QUADS);
 glColor3f(1.0, 0.5, 0.6);
 glVertex2f(-139, -100);
 glVertex2f(-139, 80);
 glVertex2f(-99, 80);
 glVertex2f(-99, -100);
 glEnd();
 glBegin(GL_QUADS);
 glColor3f(1.0, 0.1, 0.6);
 glVertex2f(-99, -100);
 glVertex2f(-99, 40);
 glVertex2f(-59, 40);
 glVertex2f(-59, -100);
 glEnd();
 glColor3f(1.0, 1.0, 0.6);
 glBegin(GL_QUADS);
 glVertex2f(121, -100);
 glVertex2f(121, 70);
 glVertex2f(161, 70);
 glVertex2f(161, -100);
 glEnd();

 glBegin(GL_QUADS);
 glColor3f(1.0, 0.5, 0.6);
 glVertex2f(161, -100);
 glVertex2f(161, 80);
 glVertex2f(201, 80);
 glVertex2f(201, -100);
 glEnd();
 glBegin(GL_QUADS);
 glColor3f(1.0, 0.1, 0.6);
glVertex2f(201, -100);
 glVertex2f(201, 40);
```

```
        glVertex2f(241, 40);
        glVertex2f(241, -100);
        glEnd();
        glColor3f(1.0, 1.0, 0.6);
        glBegin(GL_QUADS);
        glVertex2f(341, -100);
        glVertex2f(341, 70);
        glVertex2f(381, 70);
        glVertex2f(381, -100);
        glEnd();
        glBegin(GL_QUADS);
        glColor3f(1.0, 0.5, 0.6);
        glVertex2f(381, -100);
        glVertex2f(381, 80);
        glVertex2f(421, 80);
        glVertex2f(421, -100);
        glEnd();
        glBegin(GL_QUADS);
        glColor3f(1.0, 0.1, 0.6);
        glVertex2f(421, -100);
        glVertex2f(421, 40);
        glVertex2f(461, 40);
        glVertex2f(461, -100);
        glEnd();

}
void plane() //Russia
{
        glBegin(GL_QUADS);
        glColor3f(1.0, 0.5, 0.6);
        glVertex2f(-339, 300);
        glVertex2f(-339, 350);
        glVertex2f(-285, 350);
        glVertex2f(-285, 300);
        glEnd();
        glColor3f(1.0, 1.0, 0.6);
        glBegin(GL_TRIANGLES);
        glVertex2f(-339, 300);
        glVertex2f(-339, 350);
        glVertex2f(-369, 370);
        glEnd();

        glColor3f(0.7, 1.0, 0.6);
        glBegin(GL_TRIANGLES);
        glVertex2f(-285, 350);
        glVertex2f(-285, 300);
        glVertex2f(-250, 325);
        glEnd();
        glColor3f(1.0, 0.9, 0.6);
        glBegin(GL_TRIANGLES);
```

```
         glVertex2f(-330, 350);
          glVertex2f(-290, 350);
          glVertex2f(-290, 390);
          glEnd();
          glColor3f(1.0,0.9, 0.6);
          glBegin(GL_TRIANGLES);
          glVertex2f(-330, 300);
          glVertex2f(-290, 300);
          glVertex2f(-290, 260);
          glEnd();
}
void tank1()
{
          glColor3f(0.0f, 0.1f, 0.0f);
          glBegin(GL_QUADS);
          glVertex2f(-479, -429);
          glVertex2f(-479, -359);
          glVertex2f(-399, -359);
          glVertex2f(-399, -429);
          glEnd();
          glColor3f(1.0f, 1.0f, 1.0f);
          glBegin(GL_QUADS);
          glVertex2f(-459, -389);
          glVertex2f(-459, -379);
          glVertex2f(-419, -379);
          glVertex2f(-419, -389);
          glEnd();
          glColor3f(0.0f, 0.0f, 1.0f);
          glBegin(GL_QUADS);
          glVertex2f(-459, -399);
          glVertex2f(-459, -389);
          glVertex2f(-419, -389);
          glVertex2f(-419, -399);
          glEnd();
          glColor3f(1.0f, 0.0f, 0.0f);
          glBegin(GL_QUADS);
          glVertex2f(-459, -409);
          glVertex2f(-459, -399);
          glVertex2f(-419, -399);
          glVertex2f(-419, -409);
          glEnd();
          glColor3f(0.0f, 0.1f, 0.0f);

          glBegin(GL_QUADS);
          glVertex2f(-459, -359);
          glVertex2f(-459, -329);
          glVertex2f(-419, -329);
          glVertex2f(-419, -359);
          glEnd();
}
```

```
void tankgun()
{
        glBegin(GL_QUADS);
        glVertex2f(-439, -329);
        glVertex2f(-419, -289);
        glVertex2f(-414, -289);
        glVertex2f(-434, -329);
        glEnd();
}
void tank2()
{
        glColor3f(0.0f, 0.1f, 0.0f);
        glBegin(GL_QUADS);
        glVertex2f(-329, -429);
        glVertex2f(-329, -359);
        glVertex2f(-249, -359);
        glVertex2f(-249, -429);
        glEnd();
        glColor3f(1.0f, 1.0f, 1.0f);
        glBegin(GL_QUADS);
        glVertex2f(-309, -389);
        glVertex2f(-309, -379);
        glVertex2f(-269, -379);
        glVertex2f(-269, -389);
        glEnd();
        glColor3f(0.0f, 0.0f, 1.0f);
        glBegin(GL_QUADS);
        glVertex2f(-309, -399);
        glVertex2f(-309, -389);
        glVertex2f(-269, -389);
        glVertex2f(-269, -399);
        glEnd();
        glColor3f(1.0f, 0.0f, 0.0f);
        glBegin(GL_QUADS);
        glVertex2f(-309, -409);
        glVertex2f(-309, -399);
        glVertex2f(-269, -399);
        glVertex2f(-269, -409);
        glEnd();
        glColor3f(0.0f, 0.1f, 0.0f);
        glBegin(GL_QUADS);
        glVertex2f(-309, -359);
        glVertex2f(-309, -329);

        glVertex2f(-269, -329);
        glVertex2f(-269, -359);
        glEnd();
}
void tankgun2()
{
      glBegin(GL_QUADS);
       glColor3f(0.0f, 0.1f, 0.0f);
        glVertex2f(-289, -329);
        glVertex2f(-269, -289);
        glVertex2f(-264, -289);
        glVertex2f(-284, -329);
        glEnd();
}
int background()
{
        glColor3f(0.0f, 1.0f, 0.0f);//Green
        glBegin(GL_QUADS);
        glVertex2f(-600, -800);
        glVertex2f(-600, -100);
        glVertex2f(800, -100);
        glVertex2f(800, -800);
        glEnd();
}
float th = 0.0;
float trX = 0.0, trY = 0.0,x3=0.0;
void display()
{
        float x = 0, y = 0;
        glClear(GL_COLOR_BUFFER_BIT);
        glLoadIdentity();
        if (rFlag == 1) //Rotate Around origin
        {
                  trX += 0.5;
                  trY += 0.0;
        }
        background();
        buildings();
        if (rFlag == 3) {
                  x3 += 0.5;
                  y3 -= .5;
        }
        glPushMatrix();
        if (rFlag == 4) {
                  glTranslatef(-x3, 0, 0.0);
                  x3 = 0;
        }
        glTranslatef(x3, y3, 0.0);
        plane();
        glPopMatrix();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(-289.0, -329.0, 0.0);//changing origin of rotation
       glRotatef(angle, 0.0f, 0.0f, 1.0f);//rotating at origin
        glTranslatef(289.0,329.0, 0.0);//
        glPopMatrix();
```

```
 glTranslatef(trX, trY, 0.0);
 tankgun2();
  if (rFlag == 2) {
          trX = 0.0;
  }
  tank1();//russia tank
  glTranslatef(x, y, 0);
  tank2();
  glPushMatrix();
  glTranslatef(-439.0, -329.0, 0.0);//changing origin of rotation
  glRotatef(angle, 0.0f, 0.0f, 1.0f);//rotating at origin
  glTranslatef(439.0, 329.0, 0.0);//

  tankgun();
  glPopMatrix();//
  angle = angle + 0.2;
  glutPostRedisplay();
  glutSwapBuffers();
}
void myInit()
{
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-500.0, 500.0, -500.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
}
void rotateMenu(int option)
{
  rFlag = option;

}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Tank Mini Project");
  myInit();
  glutDisplayFunc(display);
  glutCreateMenu(rotateMenu);
  glutAddMenuEntry("Move Russian Tank", 1);
  glutAddMenuEntry("Reset", 2);

  glutAddMenuEntry("Move plane", 3);
  glutAddMenuEntry("Reset plane", 4);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  glutMainLoop();
}
```

CHAPTER 6

RESULTS

1.MAIN MENU

Figure 6.1 shows the start screen of the project with theMenus,Move Russian Tank,Reset,Move plane,Reset Plane

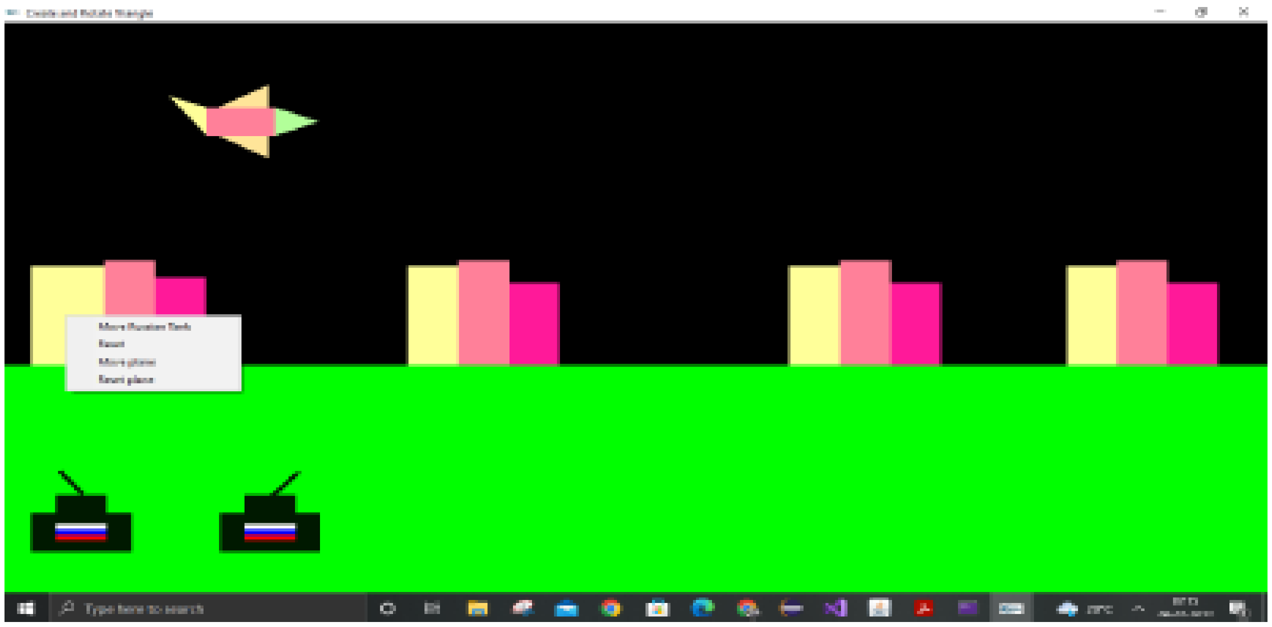

Figure 6.1 depicts Initial output of the Application

2. TRANSALTION

Fig 6.2 shows the motion of the tank by translating it from left to right.

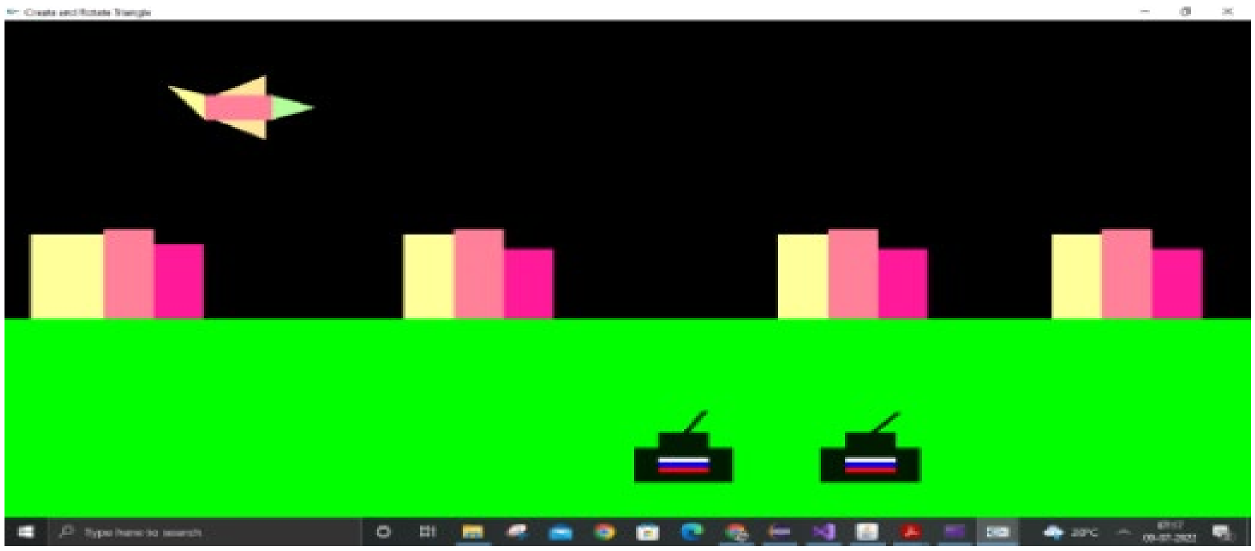

Figure 6.2 Shows the two tanks Moving

3. ROTATION

Fig 6.3 shows the gun of the tanks rotating 360 degree to check for enemies.

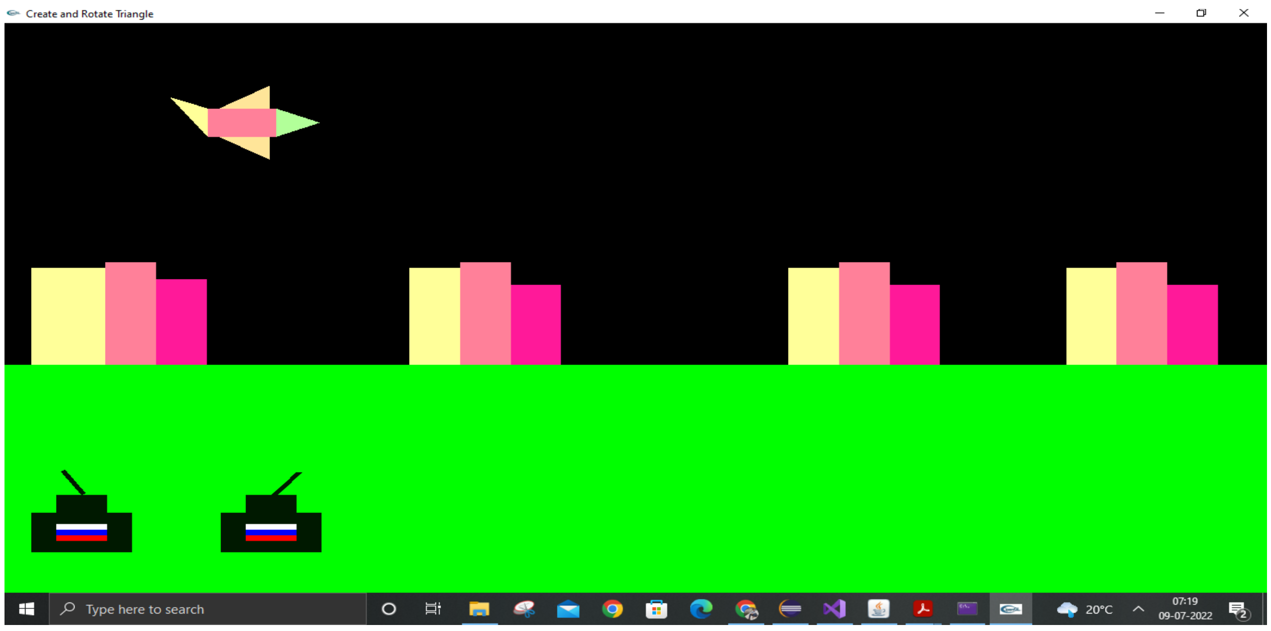

Figure 6.3 depicts Tank Gun rotating

Fig 6.4 shows the motion of the plane by translating it from left to right.

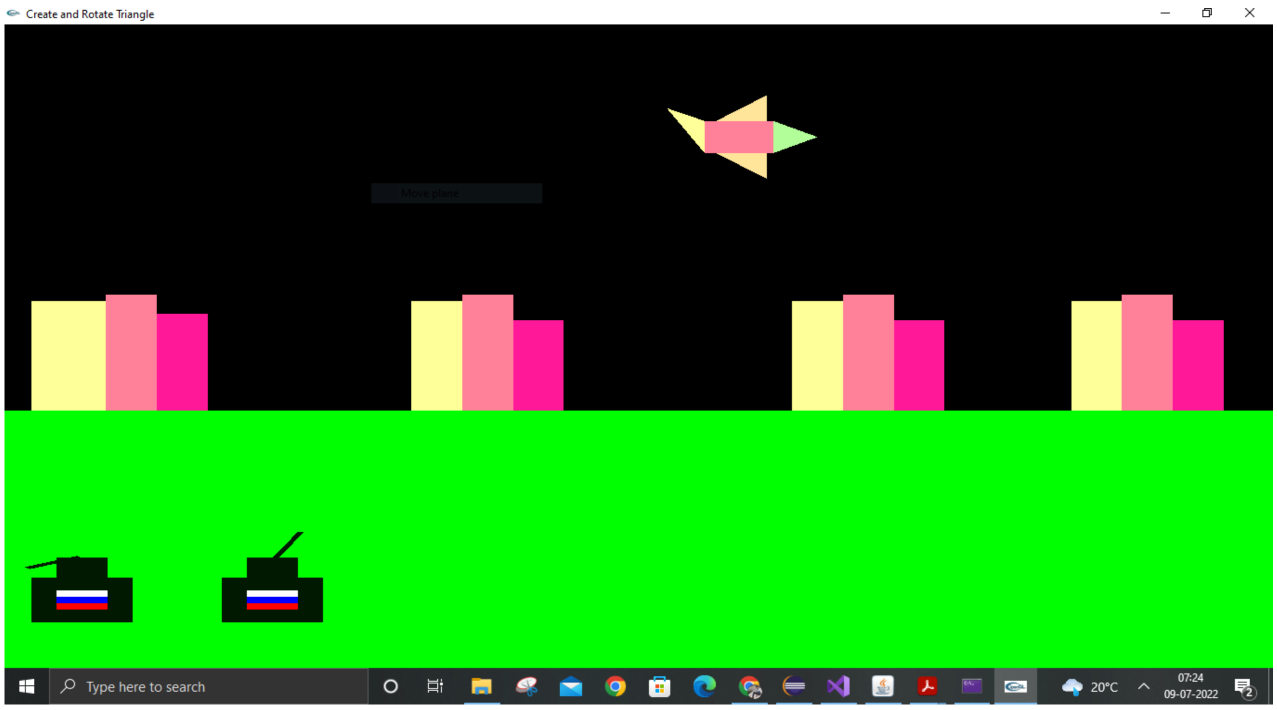

Figure 6.3 depicts Plane moving

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion

This project uses the basic computer graphics shapes like basic triangle,rectangles to build planes,building,tanks and give users a live experience of a war scenario We have used inbuilt opengl functions like glRotatef and glTranslatef to rotate and translate the artilaries of war to give real life effect.

7.2 Future enhancements

In future, the following enhancements could be done:

- Providing camera movement.
- Adding more helicopters and planes
- Showing Bomb explosions
- Showing gun shots.

BIBLIOGRAPHY

1. Edward Angel: Interactive Computer Graphics: A Top Down Approach 5th Edition, Addison – Wesley, 2008.

2. Donald Hearn and Pauline Baker: OpenGL, 3rd Edition, Pearson Education, 2004.

3. www.opengl.org

4. www.wikipedia.org

5. www.google.com