VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI - 590018

Computer Graphics
Mini Project Report
On

LIFT-OVER BRIDGE

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by
M.S.NAGAJYOTHI
(1BG18CS066)

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

2020-21

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 **LIFT-OVER BRIDGE** carried out by **Ms. NAGAJYOTHI.M.S** USN **1BG18CS066** 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 2020-21. 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 Mini Project Laboratory prescribed for the said degree.

Mrs. Akshitha Katkeri
Assistant Professor
Department of CSE
BNMIT, Bengaluru

Dr. Sahana D. Gowda
Professor & HOD
Department of CSE
BNMIT, Bengaluru

Table of Contents

List of Figures

ABSTRACT

Computers have become a powerful medium for the rapid and economical production of pictures. There is virtually no area in which graphical displays cannot be used to some advantage. Graphics provide a so natural means of communicating with a computer that they have become widespread. The fields in which Computer Graphics find their uses are many. Some of them being User Interfaces, Computer Aided Design, Office automation, Desktop Publishing, Plotting of mathematical, scientific or industrial data, Simulation, Art, Presentations, Cartography, to name a few…Here, We have tried to incorporate and present the working environment of a Lift-Over Bridge which is also known as Bascule Bridge in 2D.

The bascule bridge works with a counterweight that balances the span (leaf) while the upward swing provides clearance for boat traffic. Here, we have created a scene consisting of the bascule bridge which operates to allow a boat to pass under it while a bus waits for the leaf of the bridge to swing back into its position and then passes along the bridge after the boat has sailed across. We have provided mouse interface to start and stop animation and to exit the window. We have also included the keyboard input function to change the color of the boat.

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. Sahana D. Gowda**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

I would also like to thank **Mrs. Akshitha Katkeri,** Assistant 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.

M.S.NAGAJYOTHI

1BG18CS066

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 to not only 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

The aim of this application is to show a basic implementation of Lift-over bridge; it is a movable bridge. The application will be implemented using the C programming language and the OpenGL API. The objective of the application is to demonstrate a counter weight that continuously balances the span or “leaf” throughout and provides clearance for water traffic using the principles of Computer Graphics. The application will also include user interaction through keyboard events and menu options.

1.3 Motivation

Lift-over bridges operate using the same principle and are the most common type of movable bridge in existence as they open quickly and require relatively less energy to operate. The whole process of opening the bascules allowing a ship to pass and bringing them down again for the resumption of road traffic takes only few minutes. 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 Utility Toolkit (GLUT), later superseded by freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

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.

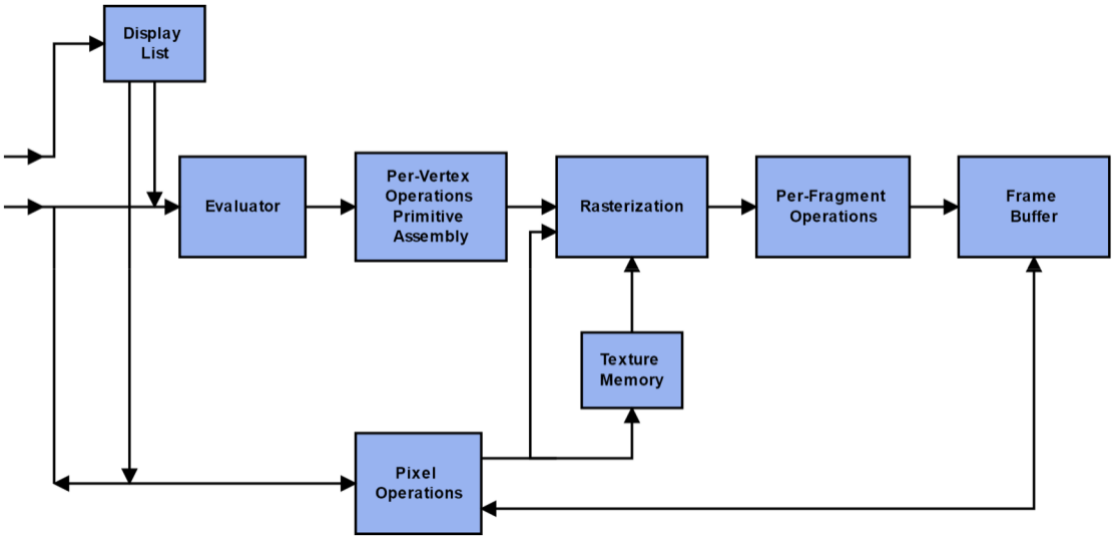

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

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 presaging the 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-Core recommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

A **vertical-lift Over Bridge** is a type of movable bridge in which a span rises vertically while remaining parallel with the deck.

The vertical lift offers several benefits over other movable bridges such as the bascule and swing-span bridge. Generally speaking they cost less to build for longer moveable spans. The counterweights in a vertical lift are only required to be equal to the weight of the deck, whereas bascule bridge counterweights must weigh several times as much as the span being lifted. As a result, heavier materials can be used in the deck, and so this type of bridge is especially suited for heavy railroad use.

The bridge allows the vehicles to move on it. When a ship approaches the bridge, a signal will be given to stop the movement of vehicles over the bridge. As soon as the vehicles stop, the cables start to lift the bascules up with the support of two towers.

Now the ship travels under the bridge without any disturbance and as soon as the ship passes the bridge area, the cables will lease down the bascules to make the way for road traffic.

A bascule is a deck that can be raised to an inclined or vertical position. Bascules either rotate in a vertical plane around a horizontal axis (similar to a seesaw, which is one meaning of the word in French) or roll back on a circular segment. This form of bridge has its antecedents in the drawbridges of medieval castles, where they served the dual purpose of spanning the moat when lowered and barricading the entry when raised. Although early examples often featured various arrangements of chains, pulleys and counterweights, the results rarely produced a truly balanced system, making it difficult to start and stop the bridge motion.

The proposed system for the application aims to simulate and demonstrate the Lift-Over Bridge using C Programming and OpenGL. OpenGL based bridge which lifts its roadway automatically whenever a ship sails towards it.

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.

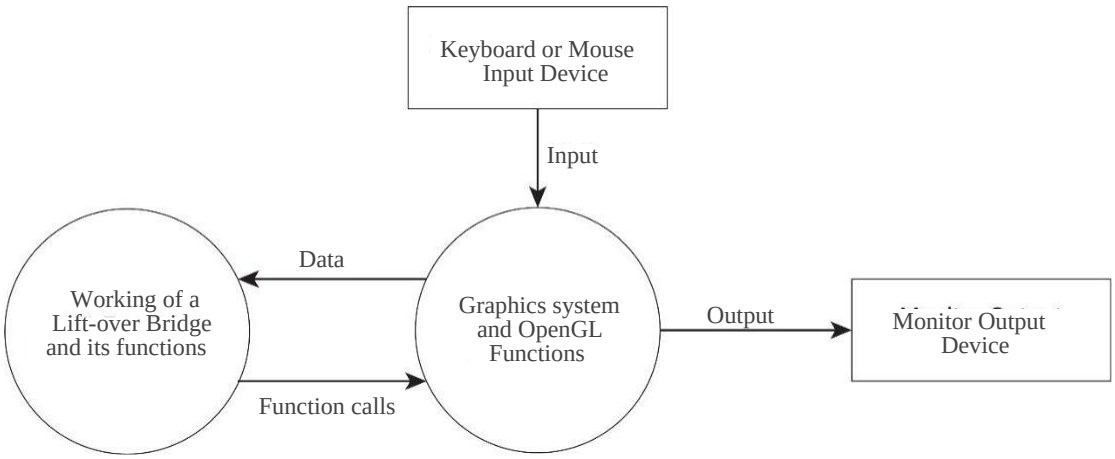

Figure 4.1 Level 0 Dataflow Diagram of the Proposed System

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.

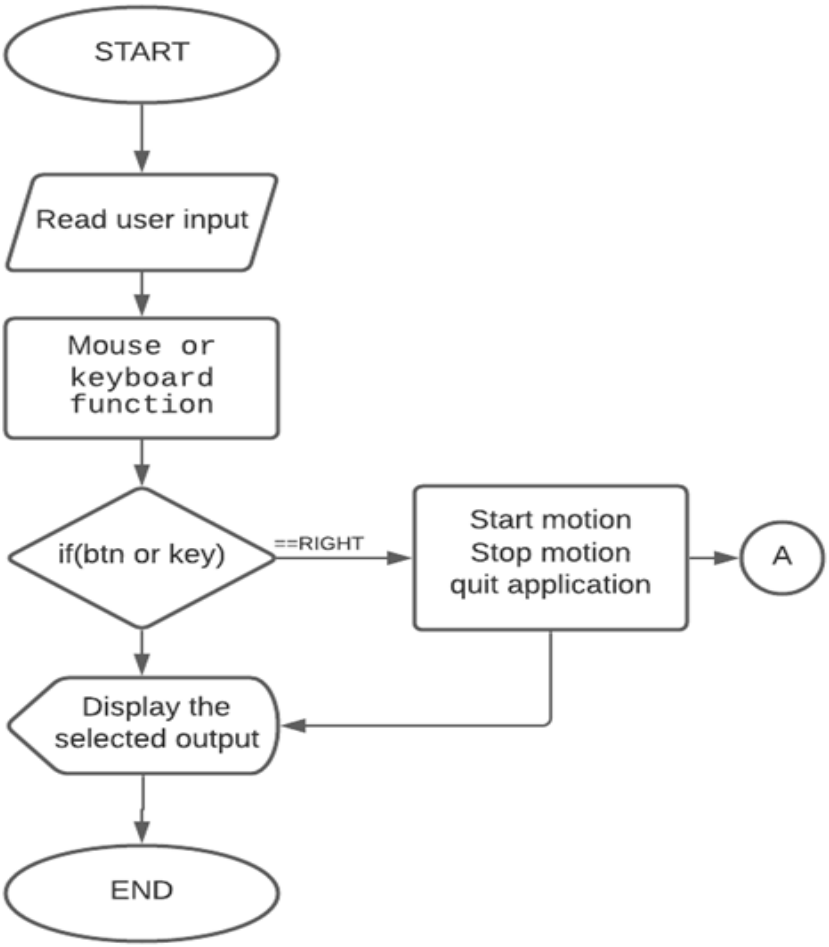

Figure 4.2 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **Void sea ():**

Since we need the blue sea, we've used the combination of green (0.5) and blue (1.0).The POLYGON function serves the purpose of covering the entire screen with blue color starting from the vertices (0, 0) to (2000, 0) and then from (2000, 1600) to (1600, 0).Then the black lines (1, 1, and 1) that represent the waves continuously translate from 0-2000 at a distance of 100 units from each other.

- **Void Bridge():**

This function represents the bridge structure in a combination of black and grey colors. They have been drawn using the GL_POLYGON function with edges in combination to represent the Top1-4 ,Strip1-4, YellowStrip1-4, Thread f & b, base1&2, Right & Left pole and the two 6-point polygons.

- **Void boat():**

This was by far the most tedious task to get the vertices of the boat/ship in the right place and orientation. We have used the following points to get them right:

(8*ship)+ (5*ship back1) + (5*ship back2) + (47*ship grill) + (4*polygon) + (4*4*table)

- **Void car/bus():**

We have translated a bus over the bridge.It was implemented by drawing a set of regular polygons and then merging them in parts to look like a bus. We used 3 sets each consisting of 4 points each to get the layout followed by a set of 16 points for the carrier. Then came the set of headlights with 2 points each followed by a set of 2 points for the horn grill and finally another couple of points for the side width.

- **void aeroplane( ):**

This function is used to draw the object aeroplane in the scene.It is created by plotting the points at the proper distances to resemble the shape of an aeroplane and then these points would be joined with the lines to make the aeroplane like image complete.

- **Void poles():**

This function created the 2 giant poles which counterweight the huge spans of bascules. They were divided into parts:

Left pole behind = 4 points, Right pole behind= 4 points, Left pole front= 4 points, Right pole front= 4 points

- **Void display():**

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

- **Void animate():**

Here, we show the movement of the bascule in short steps of 0.2 units per loop as the bascule moves from 135-149 to 1200.

- **Void myinit() and Void main_menu():**

These are the typical functions which appear in almost all programs and are described in chapter in detail.

- **Void keyboard():**

This function basically changes the color of the boat as we have implemented the suitable color codes for their respective colors. The colors that the boat can have are red, green, blue, yellow, cyan and magenta.

- **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 primary 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 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 finite time.

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

Where "mode" specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted are:

GL_MODELVIEW, GL_PROJECTION and GL_TEXTURE

- **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.

- **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.

- **glOrtho ( ):**

The function defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

- **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.

- **glutPostRedisplay()**

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

5.2.2 User Implementation

- **Function to display sea**

```
void sea()
{
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.50, 1.0);
        glVertex2f(0.0, 0.0);
        glVertex2f(2000.0, 0.0);
        glVertex2f(2000.0, 1600.0);
        glVertex2f(0.0, 1600.0);
        glEnd();

        glPushMatrix();
        glTranslatef(0, q, 0);
        glBegin(GL_LINES);
        glColor3f(1.0, 1.0, 1.0);
        for (p = 0; p < 20000; p = p + 100)
        for (s = 0; s < 20000; s = s + 100)
        glVertex2f(100.0 + s, 100.0 + p);
        glVertex2f(200.0 + s, 100.0 + p);
        glEnd();
        glPopMatrix();
}
```

- **Function to display bridge**

```
void bridge()
{
        glBegin(GL_POLYGON);
        glColor3f(0.40, 0.40, 0.40);
        glVertex2f(0.0, 900.0);
        glVertex2f(500.0, 900.0);
        glVertex2f(500.0, 1200.0);   //bridge top 1
        glVertex2f(0.0, 1200.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex2f(100.0, 1030.0);
        glVertex2f(200.0, 1030.0);
        glVertex2f(200.0, 1040.0);   //strip1
        glVertex2f(100.0, 1040.0);
        glEnd();
```

```
        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex2f(300.0, 1030.0);
        glVertex2f(400.0, 1030.0);
        glVertex2f(400.0, 1040.0);   //strip2
        glVertex2f(300.0, 1040.0);
        glEnd();


        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, .0);
        glVertex2f(0.0, 1170.0);
        glVertex2f(500.0, 1170.0);
        glVertex2f(500.0, 1175.0);   //yellow strip1
        glVertex2f(0.0, 1175.0);
        glEnd();


        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 0.0);
        glVertex2f(0.0, 920.0);
        glVertex2f(500.0, 920.0);
        glVertex2f(500.0, 930.0);   //yellow strip2
        glVertex2f(0.0, 930.0);
        glEnd();

        //   brige up

        glPushMatrix();
        glBegin(GL_POLYGON);
        glColor3f(0.46, 0.46, 0.46);
        glVertex2f(500.0, 900.0);            //bridge top 2
        //up
        glVertex2f(900.0 - k, 900.0 + n);
        glVertex2f(900.0 - k, 1200.0 + n);
        //up
        glVertex2f(500.0, 1200.0);
        glEnd();


    glBegin(GL_LINES);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(20.0, 1400.0);
        glVertex2f(900.0 - k, 900.0 + n);    //pole thread front
        glVertex2f(0.0, 1400.0);
        glVertex2f(900.0 - k, 880.0 + n);
        glEnd();
```

```
glBegin(GL_LINES);
glColor3f(0.0, 0.0, 0.0);
glVertex2f(30.0, 1550.0);
glVertex2f(900.0 - k, 1200.0 + n);   //pole thread back
glVertex2f(50.0, 1550.0);
glVertex2f(900.0 - k, 1203.0 + n);
glEnd();


glBegin(GL_POLYGON);
glColor3f(0.46, 0.46, 0.46);
glVertex2f(900.0 + k, 900.0 + n);
//up
glVertex2f(1300.0, 900.0);    // bridge top3
glVertex2f(1300.0, 1200.0);
//up
glVertex2f(900.0 + k, 1200.0 + n);
glEnd();


glBegin(GL_POLYGON);
glColor3f(0.40, 0.40, 0.40);
glVertex2f(1300.0, 900.0);
glVertex2f(2000.0, 900.0);  //bridge top 4
glVertex2f(2000.0, 1200.0);
glVertex2f(1300.0, 1200.0);
glEnd();


glBegin(GL_POLYGON);
glColor3f(1.0, 1.0, 0.0);
glVertex2f(1300.0, 1170.0);
glVertex2f(2000.0, 1170.0);
glVertex2f(2000.0, 1175.0);   //yellow strip3
glVertex2f(1300.0, 1175.0);
glEnd();

glBegin(GL_POLYGON);
glColor3f(1.0, 1.0, 0.0);
glVertex2f(1300.0, 920.0);
glVertex2f(2000.0, 920.0);
glVertex2f(2000.0, 930.0);   // yellow strip4
glVertex2f(1300.0, 930.0);
glEnd();
```

```
        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex2f(1400.0, 1030.0);
        glVertex2f(1500.0, 1030.0);
        glVertex2f(1500.0, 1040.0);   //strip3
        glVertex2f(1400.0, 1040.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex2f(1600.0, 1030.0);
        glVertex2f(1700.0, 1030.0);
        glVertex2f(1700.0, 1040.0);   //strip4
        glVertex2f(1600.0, 1040.0);
        glEnd();
}
```

- **Function to display boat**

```
void boat()
{
        glPushMatrix();
        glTranslatef(0, y, 0);
        glPushMatrix();
        glBegin(GL_POLYGON);
        glColor3f(m, j, o);
        glVertex2f(800.0, 620.0);
        glVertex2f(750.0, 200.0);     //ship
        glVertex2f(900.0, 50.0);
        glVertex2f(1000.0, 620.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.0, 0.0);     // ship back  1
        glVertex2f(750.0, 200.0);
        glVertex2f(900.0, 0.0);
        glVertex2f(900.0, 50.0);
        glVertex2f(751.0, 200.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.1, 0.1, 0.1);
        glVertex2f(901.0, 0.0);      //ship back  2
        glVertex2f(1050.0, 200.0);
        glVertex2f(901.0, 50.0);
        glEnd();
```

```
        glBegin(GL_LINES);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(900.0, 700.0);
        glVertex2f(820.0, 600.0);  //boat grill
        glVertex2f(820.0, 600.0);
        glVertex2f(800.0, 620.0);
        glVertex2f(820.0, 600.0);
        glVertex2f(770.0, 500.0);
        glVertex2f(770.0, 500.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(850.0, 400.0);//table on ship1
        glVertex2f(850.0, 350.0);
        glVertex2f(860.0, 350.0);
        glVertex2f(860.0, 400.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(920.0, 400.0);//2
        glVertex2f(930.0, 380.0);
        glVertex2f(930.0, 380.0);
        glVertex2f(920.0, 400.0);
        glEnd();


        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(950.0, 400.0);//3
        glVertex2f(950.0, 350.0);
        glVertex2f(940.0, 350.0);
        glVertex2f(940.0, 400.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(860.0, 400.0);
        glVertex2f(860.0, 380.0);
        glVertex2f(870.0, 380.0);//4
        glVertex2f(870.0, 400.0);
        glEnd();
        glPopMatrix();
        glPopMatrix();
}
```

- **Function to display car**

```
void car()
 {
        glPushMatrix();
        glTranslatef(g, 0, 0);
        glBegin(GL_POLYGON);              //  car
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(1800.0, 1050.0);
        glVertex2f(1950.0, 1050.0);
        glVertex2f(1950.0, 1150.0);
        glVertex2f(1800.0, 1150.0);
        glEnd();

        glBegin(GL_LINES);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(1780.0, 1125.0);       //head lamps
        glVertex2f(1780.0, 1135.0);
        glEnd();

        glBegin(GL_LINES);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(1800.0, 1040.0);       //side window
        glVertex2f(1928.0, 1040.0);
        glEnd();
        glPopMatrix();
   }
```

- **Function to display poles**

```
void poles()
{
        glBegin(GL_POLYGON);              //  left pole behind
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(30.0, 1200.0);
        glVertex2f(50.0, 1200.0);
        glVertex2f(50.0, 1550.0);
        glVertex2f(30.0, 1550.0);
        glEnd();

        glBegin(GL_POLYGON);              //  right pole behind
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(1725.0, 1200.0);
        glVertex2f(1745.0, 1200.0);
        glVertex2f(1745.0, 1550.0);
        glVertex2f(1725.0, 1550.0);
        glEnd();
}
```

```
glBegin(GL_POLYGON);            // street light behind 1
glColor3f(0.0, 0.0, 0.0);
glVertex2f(150.0, 1200.0);
glVertex2f(160.0, 1200.0);
glVertex2f(160.0, 1300.0);
glVertex2f(150.0, 1300.0);
glEnd();


glBegin(GL_POLYGON);            // street light behind 2
glColor3f(0.0, 0.0, 0.0);
glVertex2f(1800.0, 1200.0);
glVertex2f(1810.0, 1200.0);
glVertex2f(1810.0, 1300.0);
glVertex2f(1800.0, 1300.0);
glEnd();


glBegin(GL_POLYGON);            // street light behind 3
glColor3f(0.0, 0.0, 0.0);
glVertex2f(1300.0, 1200.0);
glVertex2f(1310.0, 1200.0);
glVertex2f(1310.0, 1300.0);
glVertex2f(1300.0, 1300.0);
glEnd();


glBegin(GL_POLYGON);            // street light behind 4
glColor3f(0.0, 0.0, 0.0);
glVertex2f(490.0, 1200.0);
glVertex2f(500.0, 1200.0);
glVertex2f(500.0, 1300.0);
glVertex2f(490.0, 1300.0);
glEnd();


glBegin(GL_POLYGON);            //  left pole front
glColor3f(0.0, 0.0, 0.0);
glVertex2f(0.0, 900.0);
glVertex2f(20.0, 900.0);
glVertex2f(20.0, 1400.0);
glVertex2f(0.0, 1400.0);
glEnd();
```

```
        glBegin(GL_POLYGON);            //  right pole front
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(1750.0, 900.0);
        glVertex2f(1770.0, 900.0);
        glVertex2f(1770.0, 1400.0);
        glVertex2f(1750.0, 1400.0);
        glEnd();


        glBegin(GL_POLYGON);            // street light front 1
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(250.0, 900.0);
        glVertex2f(260.0, 900.0);
        glVertex2f(260.0, 1000.0);
        glVertex2f(250.0, 1000.0);
        glEnd();

        glBegin(GL_POLYGON);            //  street light front 2
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(1550.0, 900.0);
        glVertex2f(1560.0, 900.0);
        glVertex2f(1560.0, 1000.0);
        glVertex2f(1550.0, 1000.0);
        glEnd();
```

- **Function to display aeroplane**

```
void aeroplane()
{
        glPushMatrix();
        glTranslatef(ae++, be++, 0);
        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex2f(0.0, 20.0);
        glVertex2f(0.0, 120.0);
        glVertex2f(270.0, 120.0);
        glVertex2f(270.0, 20.0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2f(270.0, 120.0);
        glVertex2f(320.0, 110.0);
        glVertex2f(360.0, 95.0);
        glVertex2f(390.0, 75.0);
        glVertex2f(270.0, 75.0);
        glEnd();
```

```
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(270.0, 121.0);
        glVertex2f(321.0, 111.0);
        glVertex2f(351.0, 96.0);
        glVertex2f(391.0, 76.0);
        glVertex2f(270.0, 76.0);
        glEnd();


        glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(270.0, 75.0);
        glVertex2f(390.0, 75.0);
        glVertex2f(350.0, 45.0);
        glVertex2f(310.0, 30.0);
        glVertex2f(270.0, 20.0);
        glEnd();
        glPopMatrix(); glPopMatrix(); glPopMatrix();
}
```

- **Display Function**

```
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        sea();
        bridge();
        boat();
        car();
        poles();
        aeroplane();
        glFlush();
        glutSwapBuffers();
}
```

- **Function to animate**

```
void animate()
{
        q = q - 0.5;
        y = y + 0.5;
        a = a + 0.8;
        i += 0.5;
        if ((i >= 135) && (i <= 439))
                {
                        k = k + 0.4;
                        n = n + 0.4;
                }
```

```
        if (i >= 1200 && !(k <= 0 && n <= 0))
                {
                        k = k - 0.4;
                        n = n - 0.4;
                }

                if (k <= 0)
                        g -= 1.4;
                glutPostRedisplay();
        }


        void myinit()
        {
                glClearColor(1.0, 1.0, 1.0, 1.0);
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();
                gluOrtho2D(0.0, 2000.0, 0.0, 1600.0);
        }
```

- **Keyboard Function**

```
void keyboard(unsigned char key, int x, int y)
{
        switch (key)
        {
        case 'r':m = 1.0, j = 0.0, o = 0.0;
                glutPostRedisplay();
                break;
        case 'g':m = 0.0, j = 1.0, o = 0.0;
                glutPostRedisplay();
                break;
        case 'b':m = .80, j = .50, o = 0.15;
                glutPostRedisplay();
                break;
 }
```

- **Mouse Functions**

```
void menu(int ch)
{
        switch (ch)
        {
        case 1:glutIdleFunc(animate);
                break;

        case 2:glutIdleFunc(NULL);
                break;

        case 3:exit(0);
        }
}
```

- **Main Function**

```
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(2000, 1600);
        glutInitWindowPosition(0, 0);
        glutCreateWindow(" BASCULE BRIDGE IN 2D USING OpenGL");
        glutKeyboardFunc(keyboard);
        glutCreateMenu(menu);
        glutAddMenuEntry("START MOTION", 1);
        glutAddMenuEntry("STOP MOTION", 2);
        glutAddMenuEntry("EXIT", 3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        printf("press 'r' to change the ship color to red\n");
        printf("press 'g' to change the ship color to green\n");
        printf("press 'b' to change the ship color to brown\n");
        glutDisplayFunc(display);
        myinit();
        glClearColor(1.0, 1.0, 0.0, 1.0);
        glutMainLoop();
        return 0;
 }
```

Chapter 6

RESULTS

- **Initial output**

This shows the initial stage of the application, shows the bridge with poles and street lights. It also shows a boat on the sea ready to move under the bridge when it opens .It also shows a car on the bridge which moves after the bridge closes.

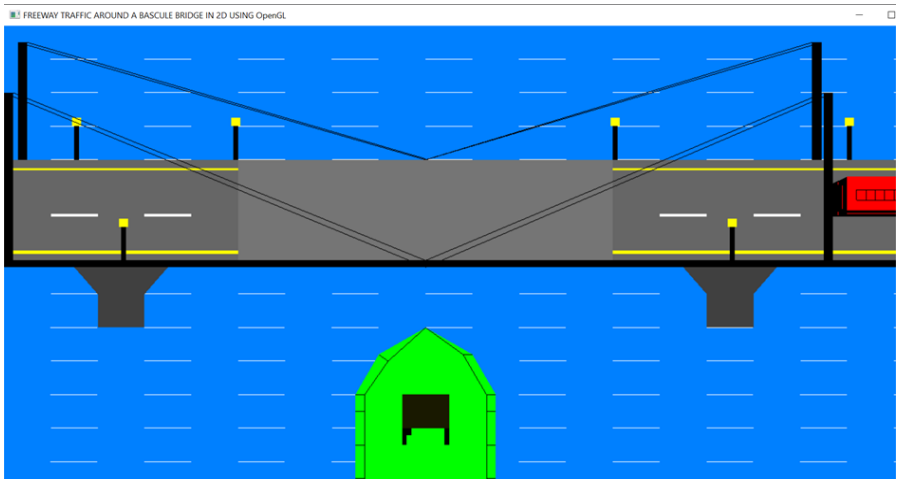

Figure 6.1 Initial output of the Application

- **Main menu(mouse interface)**

The main menu of the application, shows the mouse functions on right click for user choice. The first choice is Start motion, second is Stop motion and the last option is exit which allows the user to exit from the application.

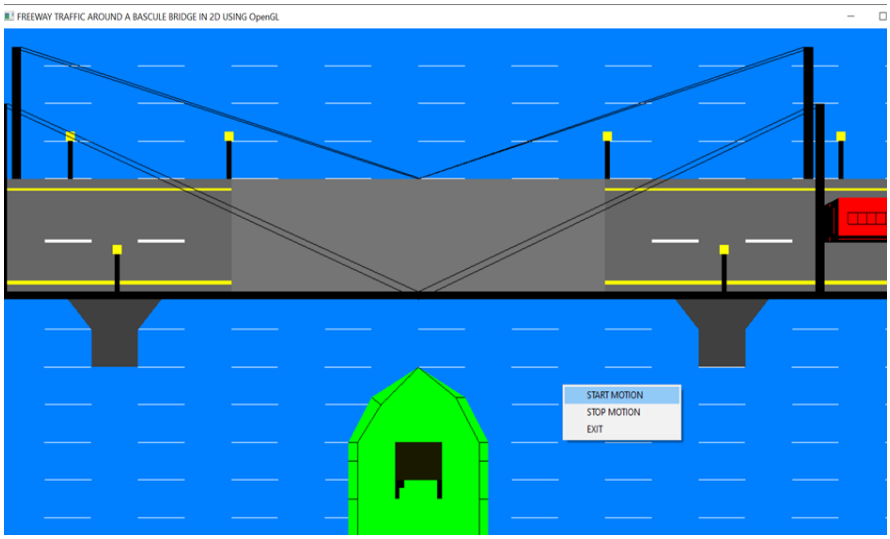

Figure 6.2 The main menu of the Application

- **Lifting up of bascules**

This shows the lifting up of bascules and movement of vehicles. While the bascule is being lifted the aeroplane quickly passes by over the bridge and then the ship passes by on the sea after the bascules are lifted completely.

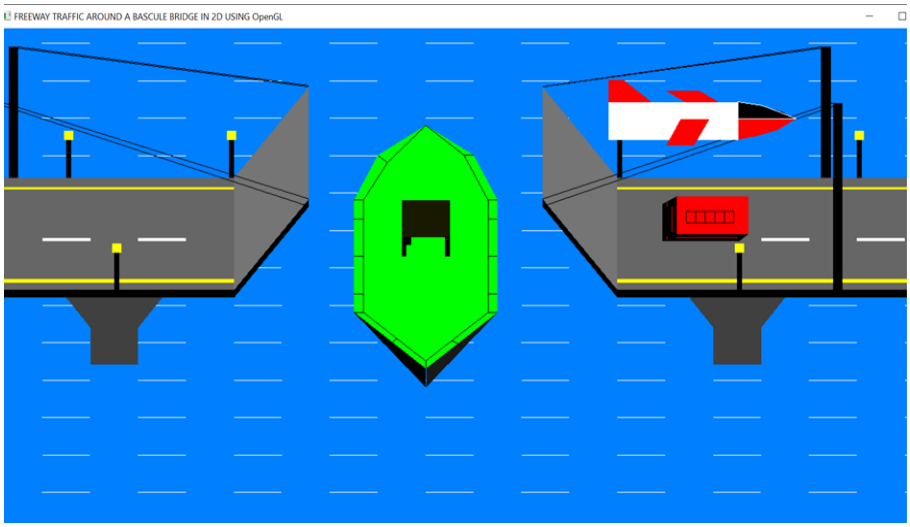

Figure 6.3 Lifting up of bascules

- **Changing the colors of boat**

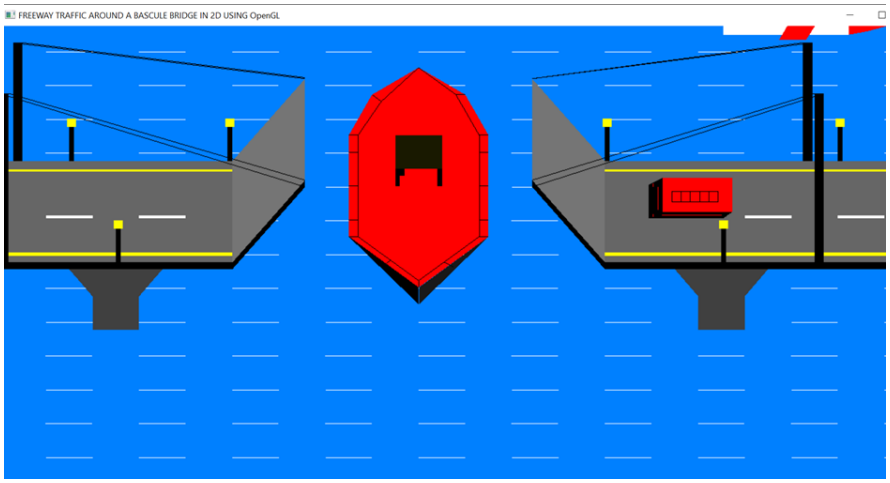

Figure 6.4 Changing boat colours

With the keyboard functions we can change the colour of boat in 6 different colors as per the choice of the user through keyboard interactions. The 6 primitive colours which can be used are red, green, blue, brown, cyan & magenta. These include both additive and primitive colours.

- **Lifting down of bascules**

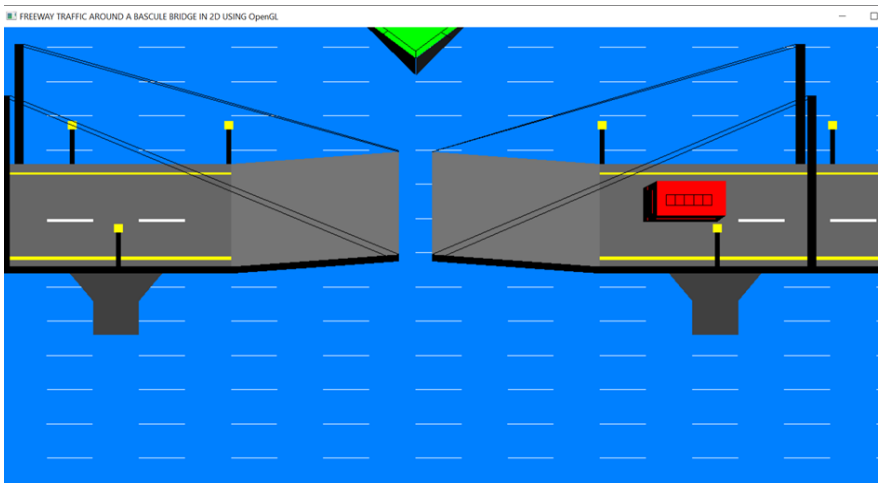

Figure 6.5 Lifting down of bascules

Figure 6.5 shows lifting down of bridge. Here the bascule closes as soon as it gets a signal after the ship has completely passed under it and moved further. After the bascules are closed it makes way for vehicles to pass over the bridge.

- **Vehicle crossing over the bridge(final output)**

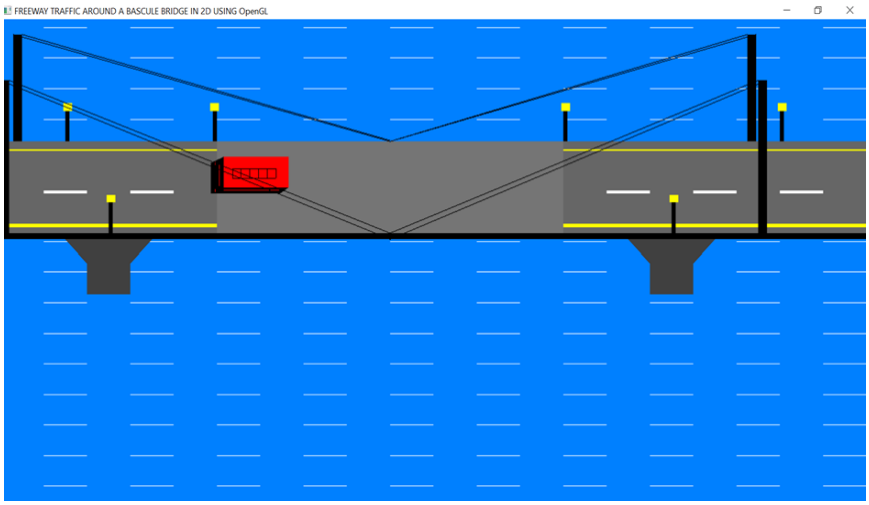

Figure 6.6 Car moving over the bridge

This is the final stage before exiting from the application. Here after the bascules are closed completely vehicles are allowed to move on the bridge. After the vehicle reaches the end of the bridge we can right click the mouse button n click on exit .

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

The working of a lift-over bridge demonstrates the opening of bascules when a ship approaches the bridge and closes bascules allowing the movement of vehicles over the bridge. During the course of building this application we utilised various OpenGL API Functions and variables that made it easier for us to visualize the applications working along with the working of the functions that were used.

Finally I conclude by saying that this program is completed successfully using OpenGL and ready to be demonstrated .

In the future this application can be further enhanced by adding more functionality to show more operations on working of a Lift-over Bridge like including sounds of sea, boat, bus and bridge movement can be incorporated. Support for advanced 3D representation of the entire scenario.

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] Web references: www.opengl.org

www.wikipedia.org

www.google.com