VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

Mini Project Report on

"BASCULE BRIDGE"

Submitted in the partial fulfillment for the requirements of Computer Graphics & Visualization Laboratory of 6th semester CSE requirement in the form of the Mini Project work

S Satish Kumar Reddy	USN: 1BY15CS132
Syed Arshad Basha	USN: 1BY15CS227
Chirag V	USN: 1BY18CS228

Under the guidance of

Ms. Ambika G.N
Assistant Professor, CSE, BMSIT&M

Mr. Shankar R
Assistant Professor, CSE, BMSIT&M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.

2019-2020

BMS INSTITUTE OF TECHNOLOGY &MANAGEMENT

YELAHANKA, BENGALURU – 560064

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project work entitled **"SELECTIVE ARQ PROTOCOL"**is a bonafide work carried out by **Divyashree G (1BY15CS025) and ChandraShekar S (1BY15CS020)** in partial fulfillment for *Mini Project* during the year 2sss019-2020. It is hereby certified that this project covers the concepts of *Computer Graphics & Visualization*. It is also certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report.

Signature of the
Guide with date
Mr. SHANKAR R
Assistant Professor
CSE, BMSIT&M

Signature of the
Guide with date
Ms. AMBIKA G.N
Assistant Professor
CSE, BMSIT&M

INSTITUTE VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

INSTITUTE MISSION

Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface.

DEPARTMENT VISION

To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

DEPARTMENT MISSION

Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

PROGRAM EDUCATIONAL OBJECTIVES

1. Lead a successful career by designing, analyzing and solving various problems in the field of Computer Science & Engineering.
2. Pursue higher studies for enduring edification.
3. Exhibit professional and team building attitude along with effective communication.
4. Identify and provide solutions for sustainable environmental development.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and every one who has helped us make this project a success.

We heartily thank our Principal, Dr. MOHAN BABU G N, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We heartily thank our Professor and Head of the Department, Dr. Bhuvaneshwari C M, Department of Computer Science and Engineering, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We gracefully thank our Project Guide, Mr. Shankar R, Assistant Professor, Department of Computer Science and Engineering for his intangible support and for being constant backbone for our project.

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly, we thank our parents and friends for the support and encouragement given throughout in completing this precious work successfully.

S SATISH KUMAR REDDY (1BY18CS132)
S ARSHAD BASHA (1BY18CS227)
CHIRAG V (1BY18CS228)

ABSTRACT

Automatic Repeat request (ARQ), is an error-control method for data transmission that uses acknowledgements (messages sent by the receiver indicating that it has correctly received a data frame or packet) and timeouts (specified periods of time allowed to elapse before an acknowledgment is to be received) to achieve reliable data transmission over an unreliable service. If the sender does not receive an acknowledgment before the timeout, it usually re-transmits the frame/packet until the sender receives an acknowledgment or exceeds a predefined number of re-transmissions.

The types of ARQ protocols include Stop-and-wait ARQ, Go-Back-N ARQ, and Selective Repeat ARQ / Selective Reject.

- Stop-and-wait ARQ,is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order. It is the simplest automatic repeat-request (ARQ) mechanism.
- Go-Back-N ARQ is a specific instance of the automatic repeat request (ARQ) protocol, in which the sending process continues to send a number of frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver.
- Selective Repeat ARQ / Selective Reject ARQ is a specific instance of the Automatic Repeat-Request (ARQ) protocol used to solve sequence number dilemma in communications.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 COMPUTER GRAPHICS

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer hardware and software. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. 2D computer graphics are digital images—mostly from two-dimensional models, such as 2D geometric models, text (vector array), and 2D data. 3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering images.

The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch screen. Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patter-recognition abilities allow us to perceive and process data rapidly and efficiently. In many designs, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

1.2 OPEN GL

OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. Information regarding OpenGL is all over the Web and in print. It is impossible to exhaustively list all sources of OpenGL information. OpenGL programs are typically written in C and C++. One can also program OpenGL from Delphi (a Pascal-like language), Basic, Fortran, Ada, and other languages. To compile and link OpenGL programs, one will need OpenGL header files. To run OpenGL programs, one may need shared or dynamically loaded OpenGL libraries, or a vendor-specific OpenGL Installable Client Driver (ICD).

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn).

1.3 GLUT

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres, and cylinders. GLUT even has some limited support for creating pop-up menus. The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross platform) and to make learning OpenGL easier. All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

CHAPTER 2

LITERATURE SURVEY

CG (Computer graphics) started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computer themselves. It includes the creation, storage, and manipulation of models and images of objects. These models include physical, mathematical, engineering, architectural, and even conceptual or abstract structures, natural phenomena, and so on. Computer Graphics today is largely interactive- the user controls the contents, structure, and appearance of objects and their displayed images by using input devices, such as keyboard, mouse or touch sensitive panel on the screen. Bitmap graphics is used for user-computer interaction. A Bitmap is an ones and zeros representation of points (pixels, short for picture elements ‘) on the screen. Bitmap graphics provide easy-to-use and inexpensive graphics-based applications.

The concept of desktop is a popular metaphor for organizing screen space. By means of a window manager, the user can create, position, and resize rectangular screen areas, called windows, that acted as virtual graphics terminals, each running an application. This allowed users to switch among multiple activities just by pointing at the desired window, typically with the mouse. Graphics provides one of the most natural means of communicating with the computer, since our highly developed 2D and 3D pattern – recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. In many design, implementation, and construction processes, the information pictures can give is virtually indispensable.

Computer graphics is the creation and manipulation of pictures with the aid of computers. It is divided into two broad classes:

- Non-Interactive Graphics.
- Interactive Graphics.

2.1 NON – INTERACTIVE GRAPHICS

This is a type of graphics where observer has no control over the pictures produced on the screen. It is also called as Passive graphics.

2.2 INTERACTIVE GRAPHICS

This is the type of computer graphics in which the user can control the pictures produced. It involves two-way communication between user and computer. The computer upon receiving signal from the input device can modify the displayed picture appropriately. To the user it appears that the picture changes instantaneously in response to his commands. The following fig. shows the basic graphics system.

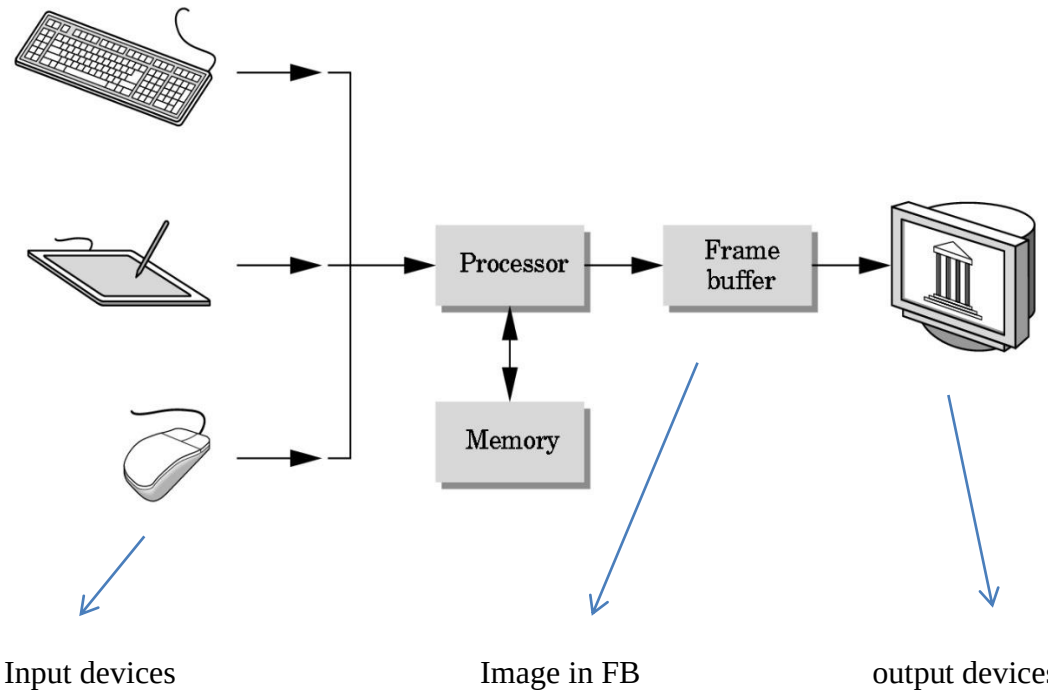


Fig 2.1: Basic Graphics System

Chapter 3

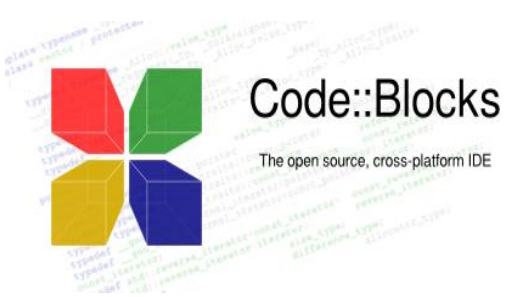
REQUIREMENT SPECIFICATION

3.1 HARDWARE REQUIREMENTS

- Processor: INTEL / AMD
- Main memory: 2 GB RAM (Min.)
- Hard Disk: Built-in
- Keyboard: QWERTY
- Mouse: 2 or 3 Button mouse
- Monitor: 1024 x 768

3.2 SOFTWARE REQUIREMENTS

- Programming language - C/C++ using OpenGL
- Operating system – Windows 10 / Ubuntu
- Compiler – C/C++ Compiler
- IDE – Code blocks
- Functional Requirement - <GL/glut.h>



3.3 Functional Requirements-

The whole project is divided into many small parts known as functions and these functions would take care of implementing particular parts of this project which makes the programming easy and effective. Each function takes the responsibility of designing the assigned parts hence we combined all the functions at a particular stage to get the final required design.

The functions that are used to implement the Lift-Over Bridge are:

- **Void sea () -** This function depicts the sea by drawing some horizontal lines on the window and translating them in the direction opposite to that of the ship. This function makes use of the OpenGL functions to define the window size, length of the horizontal lines, to provide the color for sea and make them to translate in required direction.
- **Void bridge () -** This function depicts the bridge in the scene. This function designs the bridge strip by strip. It defines the structure of the bridge. This function also designs the pole threads.
- **Void boat ()** - This function depicts the ship in the scene. A ship would be created by plotting the points at the proper distances to resemble a ship and then these points are joined with lines to make the ship image complete. This function use the OpenGL inbuilt functions especially for plotting and joining the points.
- **Void car ()** - This function used to draw the object bus in the scene. It is created by plotting the points at the proper distances to resemble the shape of a bus and then these points would be joined with the lines to make the bus like image complete.
- **Void pole () -** This function used to draw the poles on both sides of the bridge. On each side, two pole are drawn using the OpenGL inbuilt functions.
- **Void keyboard** (unsigned char key, int x, int y) - This function used to provide the keyboard interface to the user. Using this function, we can change the color of the boat or ship by pressing the corresponding keys. For example, if we press the key 'Y' color of the boat changes to yellow, similarly if we press the button 'B' color changes to Blue and so on.

Chapter 4

ALGORITHM DESIGN AND ANALYSIS

4.1 Pseudo Code:

The main algorithm for the Lift-Over Bridge can be written as below:

- Initially define the void sea () function and keep it running throughout the course of the program. ¬ Initialize the void bridge (), void pole () and void thread () functions.
- Initialize the keyboard and mouse interface functions.
- Keep the above scenery in place on the window and wait for the user to press the ―start animation button‖.
- As soon as the above step is performed, the following actions should be performed simultaneously:

  - The stream should start flowing continuously.
  - A bus (vehicle) which is passing by the bridge should halt as the span of the bridge opens up in a slow and steady movement in order to facilitate the movement of the boat underneath.
  - The boat should steadily sail across underneath the bridge while the bus waits for the bridge to close back into position. (When the user presses the respective keys on the keyboard, the color of the boat should change accordingly).
  - As soon as the leaves of the bridge close, the bus should pass along the bridge in a uniform and steady manner.

- During any time of the execution of the program, if the user presses the Stop Animation‖ button, the execution should halt (pause) and resume back if Start Animation‖ is again pressed.
- During the entire course of execution, if the user presses the Exit button, the window should exit immediately

4.2 Analysis:

The functions and their algorithms are as follows:

- **Void sea ():** Since we need the blue sea, we ‘ve use 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 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 points form the basic ship design. We had to use 5 points each to get the back of the ship to appear elevated and give it a realistic feel. And then we used 47 points to get the grills/boundaries of the ship in the desired order. We placed a polygon inside the boat which took another 4 points along with 16 points for the table in the boat. So finally, it took a whopping 85 points to get the ship to appear the way it is… (8*ship) + (5*ship back1) + (5*ship back2) + (47*ship grill) + (4*polygon) + (4*4*table)

- **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 car/bus ():** We have translated a bus over the bridge. Constructing the bus was again a challenge but was easier to implement once we had got the boat done. 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 windows.

- **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 Right pole thread front= 2 points static + 2 points dynamic Right pole thread back= 2 points static + 2 points dynamic

- **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 chapter3 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 been 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

Additive color:

- Form a color by adding amounts of three primaries
- CRTs, projection systems, positive film
- Primaries are Red (R), Green (G), Blue (B)

Chapter 5

IMPLEMENTATION

The Lift Over Bridge can be implemented using some of the OpenGL inbuilt functions along with some user defined functions. The inbuilt OpenGL functions that are used mentioned under the FUNCTIONS USED category. The user defined functions are mentioned under USER DEFINED FUNCTIONS category.

5.1 Open GL Functions Used –

- **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. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window for relative coordinates when the key gets pressed. Prototype is as given below:

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

When a new window is created, no keyboard callback is initially registered, and the ASCII keystrokes that are within the output window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks

- **Void GLflush():**

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. 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
The initial value is GL_MODELVIEW.
The function GlMatrixMode sets the current matrix mode. Mode can assume one of these values:
GL_MODELVIEW: Applies matrix operations to the model view matrix stack.
GL_PROJECTION: Applies matrix operations to the projection matrix stack.

- **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.
Viewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let (xnd, ynd) be normalized device coordinates. Then the window coordinates (xw, yw) are computed as follows:

xw =(xnd +1 width/2 + x

yw = (ynd + 1) height/2 + y

- **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. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. GlutInit also processes command line options, but the specific options parse are window system dependent

- **glOrtho ( ):**

Syntax: void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

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 USER DEFINED FUNCTIONS:

- **void sea ():**

  This function depicts the sea by drawing some horizontal lines on the window and translating them in the direction opposite to that of the ship. This function makes use of the OpenGL functions to define the window size, length of the horizontal lines, to provide the color for sea and make them to translate in required direction.

- **Void bridge ():**

  This function depicts the bridge in the scene. This function designs the bridge strip by strip. This function first defines the top part of the bridge by making use of OpenGL functions. Similarly, it designs the bottom strip and joins the top and bottom strips by drawing two side strips. This function not only draws the bridge strips but also design the pole threads.

- **Void boat ():**

  This function depicts the ship in the scene. A ship would be created by plotting the points at the proper distances to resemble the shape of a ship and then these points would be joined with the lines to make the ship like image complete. This function use the OpenGL inbuilt functions especially for plotting the points and then join in the proper manner

- **Void car ():**

  This function used to draw the object bus in the scene. A bus or car would be created by plotting the points at the proper distances to resemble the shape of a bus and then these points would be joined with the lines to make the bus like image complete.

- **Void pole ():**

  This function used to draw the poles on the both sides of the bridge. Each side two pole are drawn using the OpenGL inbuilt functions.

- **Void animate ():**

  This function used to give the step size of translation for each object in the scene.

- **Void main menu (int ch):**

  This function would provide the menu that consists following options:

  a) START ANIMATION

  b) STOP ANIMATION

  c) QUIT

  Hence it provides the mouse interface to user.

- **Void keyboard (unsigned char key, int y):**

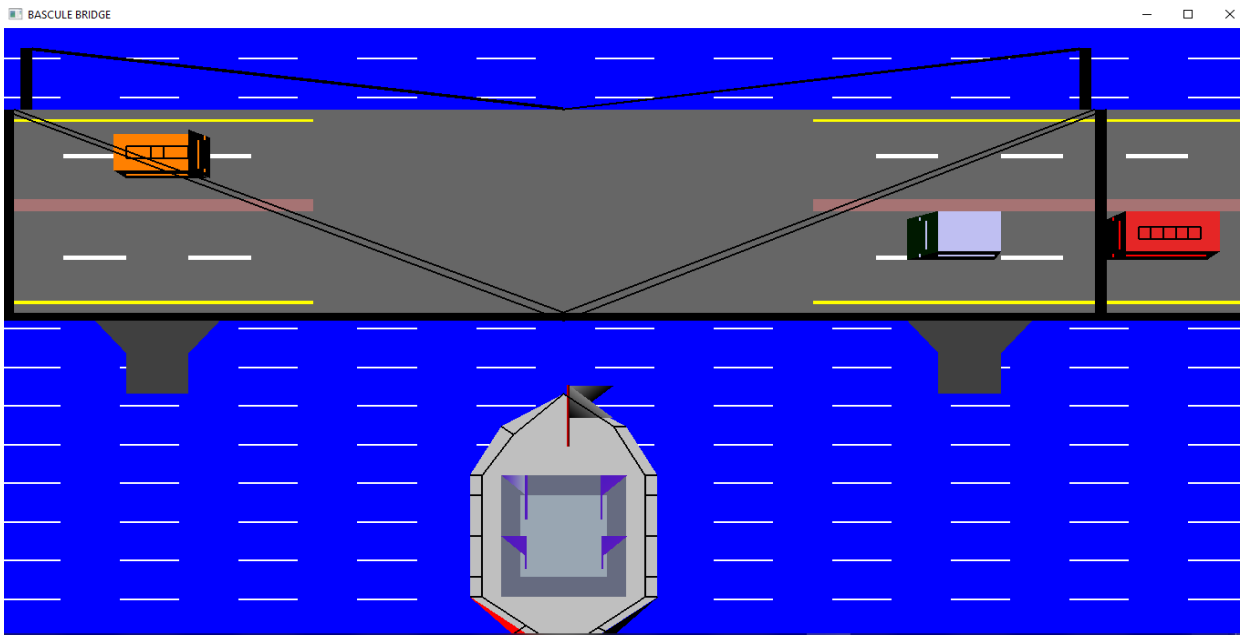
  This function used to provide the keyboard interface to the user. Using this function, we can change the color of the boat or ship by pressing the corresponding keys. For example, if we press the key 'Y' color of the boat changes to yellow, similarly if we press the button 'B' color changes to red and so on.

- **Void display(void):**

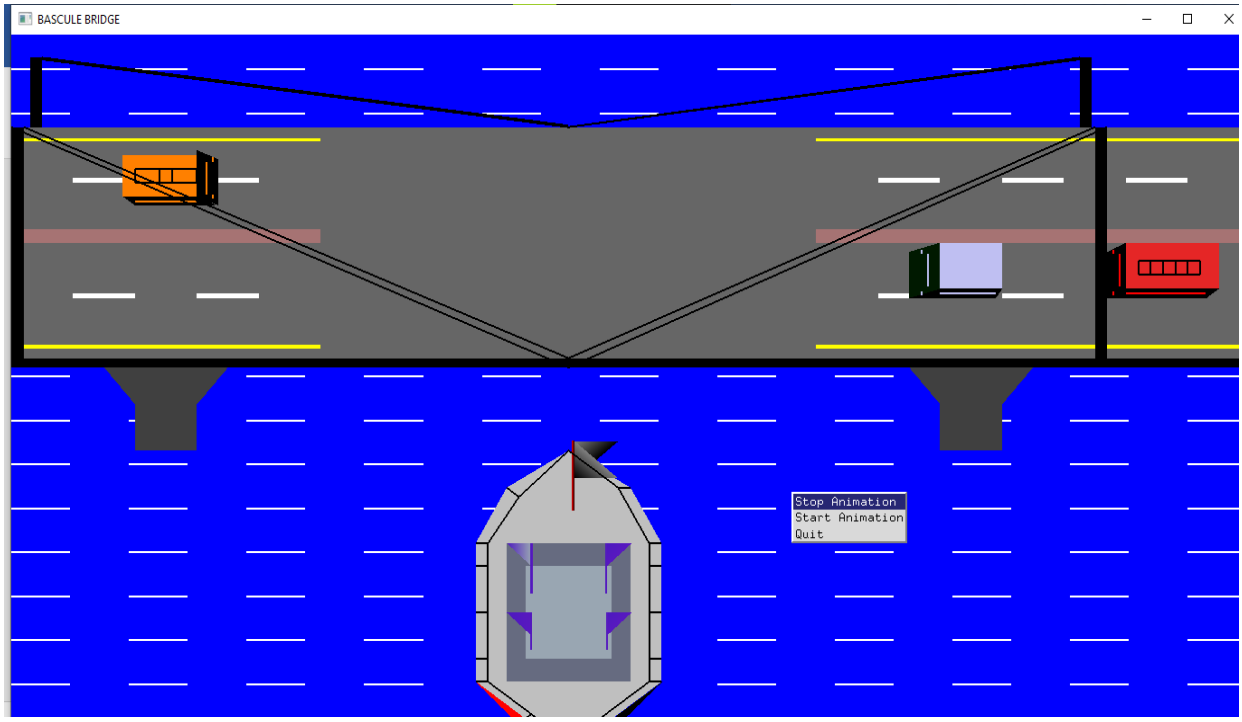
  In this function first we should print the instructions that we would be displayed on the pop-up window.

5.3 SNAPSHOTS

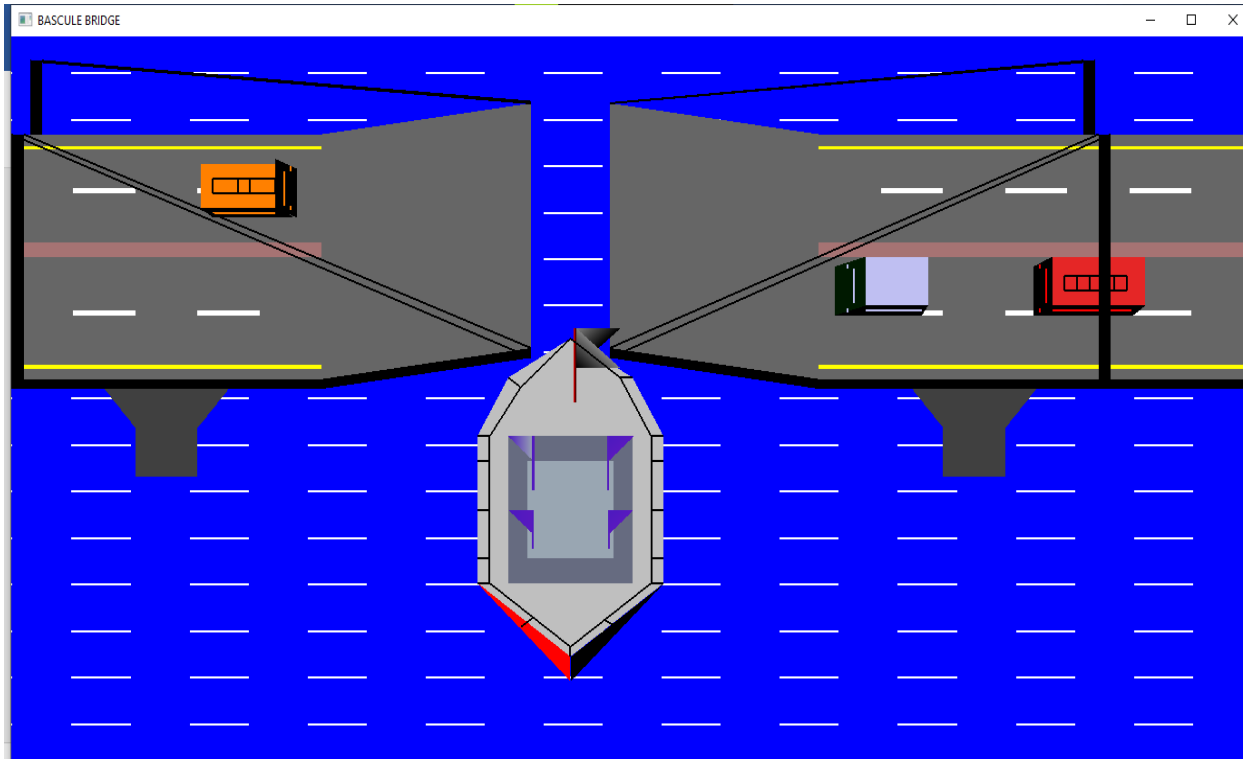
• Snap shot of Initial stage of the Lift over bridge:



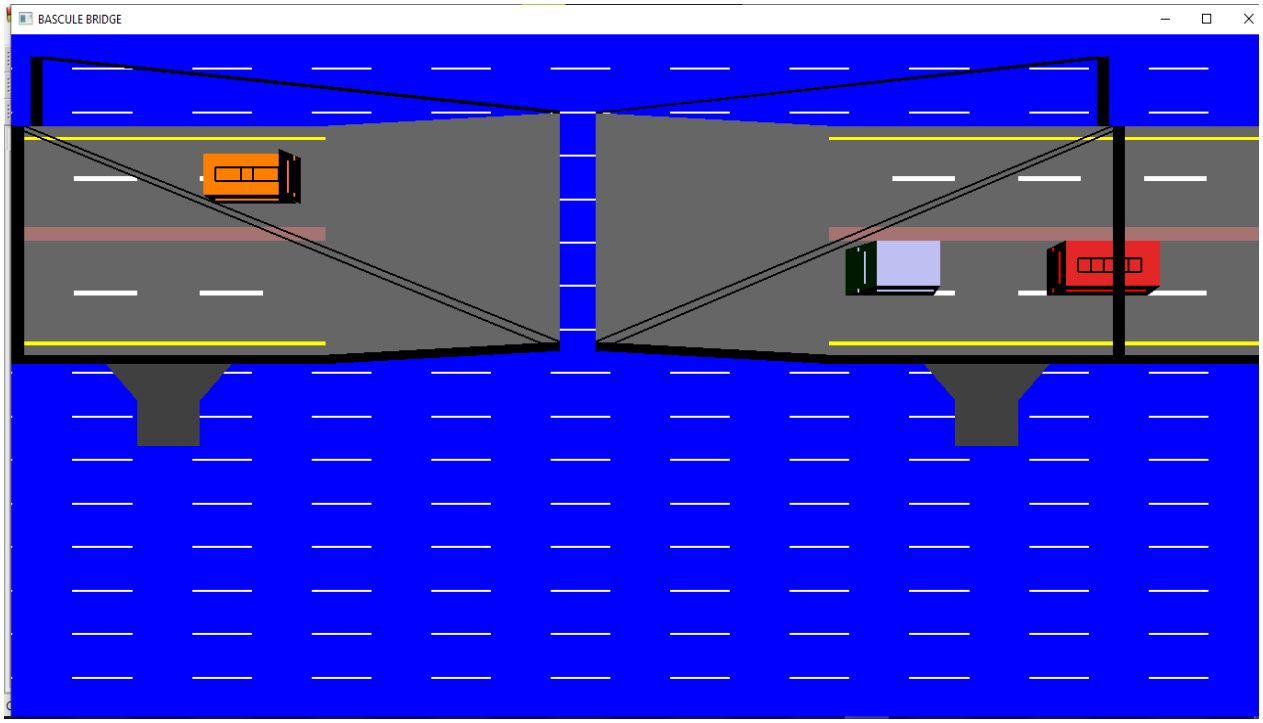
• Snap Shot Showing Main Menu (mouse interface):



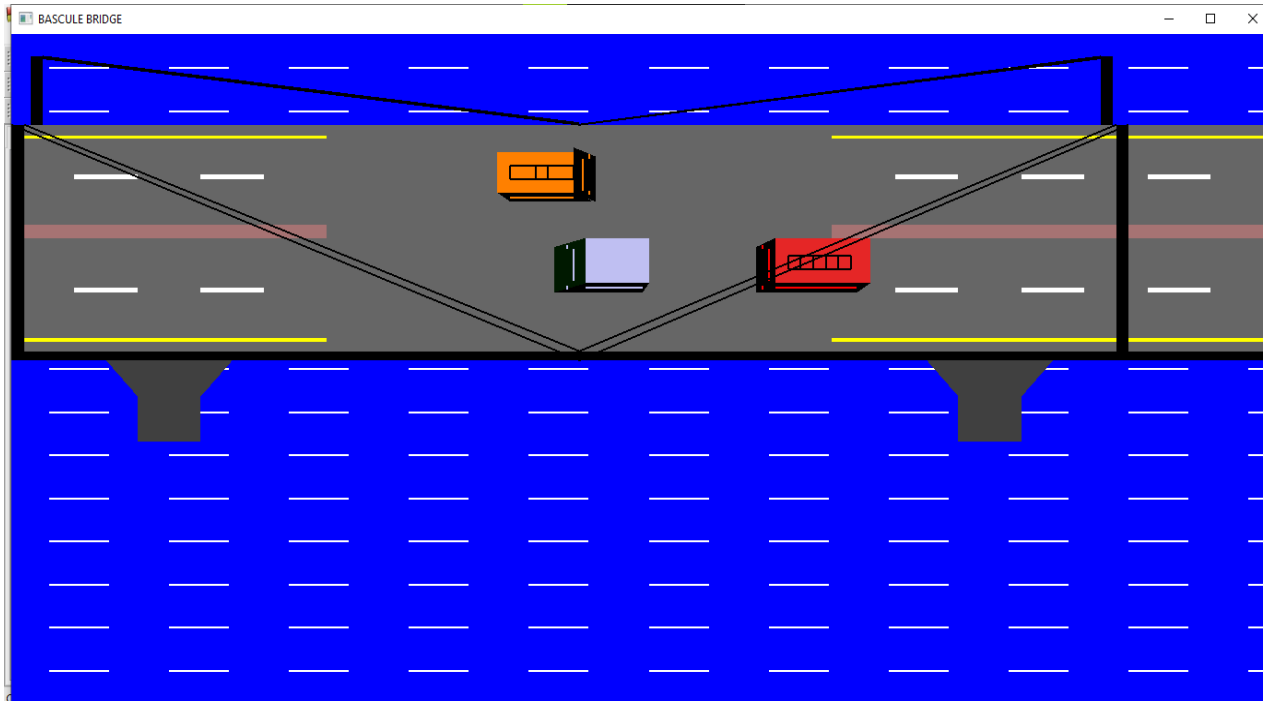
• Snapshot shows the lifting up the bascules:



- Snap Shot Shows the Bridge moving down:



- Snap Shot Shows the Vehicle Crossing Over the Bridge:



Chapter 6

CONCLUSIONS AND FUTURE SCOPE

6.1 General Constraints:

- As the software is being built to run on Ubuntu platform, which gives access to limited conventional memory, the efficient use of the memory is very important.
- As the program needs to be run even on low-end machines the code should be efficient and optimal with the minimal redundancies.
- Needless to say, the computation of algorithms should also be robust and fast.
- It is built assuming that the standard output device (monitor) supports colors.

6.2 Assumptions and Dependencies:

- One of the assumptions made in the program is that the required libraries like GL, GLU and glut have been included.
- The user's system is required to have the C compiler of the appropriate version.
- The system is also expected to have a keyboard and mouse connected since we provide the inputs via these devices.

6.3 Further Enhancements:

- The following are some of the features that can be included in the revised versions of this code are:
- Sounds of sea, boat, bus and bridge movement can be incorporated.
- Support for different types of vehicles all moving simultaneously on bridge.
- Support for advanced 3D representation of the entire scenario.
- Support for transparency of layers and originality

BIBLIOGRAPHY

Book References:

1. Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd / 4th Edition, Pearson Education,2011
2. Edward Angel: Interactive Computer Graphics- A Top-Down approach with OpenGL, 5th edition. Pearson Education, 2008
3. Kelvin Sung, Peter Shirley, steven Baer: Interactive Computer Graphics, concepts and applications, Cengage Learning
4. M M Raikar & Shreedhara K S Computer Graphics using OpenGL, Cengage publication

Web References:

- www.opengl.org
- www.codeblocks.org)
- www.sourcecode.com
- www.pearsoned.co.in

APPENDICES

9.1 Source Code:

```
#include<stdio.h>
#include<GL/glut.h>
#define SPACEBAR 32

////////////////////////////////////// Declaration of global variables  ////////////////////////
float y=0,ang=0,i=0,k=0,n=0;

float a=900,b=880,c=900,d=900,p,q=0,s;

float g=0,z=0,w=0;  // car  translate indicator

float m=.80,j=.50,o=.15;

////////////////////////////// sea function to display river  ////////////////////////////////
void sea()
        {
        glColor3f(0.0,0.0,0.0);
        glBegin(GL_POLYGON);
        glColor3f(0.0,0.0,1.0);
        glVertex2f(0.0,0.0);
        glVertex2f(2000.0,0.0);
        glVertex2f(2000.0,1600.0);
        glVertex2f(0.0,1600.0);
        glEnd();

        glPushMatrix();
        glTranslatef(q,0,0);

        glBegin(GL_LINES);
        glColor3f(1.0,1.0,1.0);
        for(p=0;p<21000;p=p+95)
        for(s=0;s<21000;s=s+95)
        glVertex2f(0.0+s,100.0+p);
        glVertex2f(0.0+s,100.0+p);


        glEnd();
        glPopMatrix();
        }

////////////////////////////////////// Bridge function ////////////////////////////////////
```

```
void bridge()
{
	glBegin(GL_POLYGON);
		glColor3f(0.40,0.40,0.40);
		glVertex2f(0.0,900.0);
		glVertex2f(500.0,900.0);
		glVertex2f(500.0,1400.0);   //bridge top 1
		glVertex2f(0.0,1400.0);
	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(100.0,1280.0);
		glVertex2f(200.0,1280.0);
		glVertex2f(200.0,1290.0);   //strip1 above 1st
		glVertex2f(100.0,1290.0);
	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(1.0,1.0,1.0);
		glVertex2f(300.0,1280.0);
		glVertex2f(400.0,1280.0);
		glVertex2f(400.0,1290.0);   //strip2 above 2nd
		glVertex2f(300.0,1290.0);
	glEnd();

	glBegin(GL_POLYGON);
		glColor3f(1.0,1.0,0.0);
		glVertex2f(0.0,1370.0);
		glVertex2f(500.0,1370.0);
		glVertex2f(500.0,1375.0);   //yellow strip1
		glVertex2f(0.0,1375.0);
	glEnd();

	glBegin(GL_POLYGON);
```

```
        glColor3f(0.65,0.45,0.45);
        glVertex2f(0.0,1150.0);
        glVertex2f(500.0,1150.0);
        glVertex2f(500.0,1180.0);   //divider 1
        glVertex2f(0.0,1180.0);
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(0.40,0.40,0.40);
        glVertex2f(500.0,900.0);          //bridge top 2
        //up
        glVertex2f(900.0-k,900.0+n);
        glVertex2f(900.0-k,1400.0+n);
        //up
        glVertex2f(500.0,1400.0);
glEnd();


glBegin(GL_LINES);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(30.0,1550.0);
        glVertex2f(900.0-k,1400.0+n);   //pole thread back
        glVertex2f(50.0,1550.0);
        glVertex2f(900.0-k,1403.0+n);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(500.0,880.0);
        glVertex2f(900.0-k,880.0+n); //base1
        glVertex2f(900.0-k,900.0+n);
        glVertex2f(500.0,900.0);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(0.40,0.40,0.40);
        glVertex2f(900.0+k,900.0+n);
        //up
```

```
        glVertex2f(1300.0,900.0);   // bridge top3
        glVertex2f(1300.0,1400.0);
        //up
        glVertex2f(900.0+k,1400.0+n);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(900.0+k,880.0+n);
        glVertex2f(1300.0,880.0);          // base 2
        glVertex2f(1300.0,900.0);
        glVertex2f(900.0+k,900.0+n);
glEnd();
glPopMatrix();

glBegin(GL_POLYGON);
        glColor3f(0.40,0.40,0.40);
        glVertex2f(1300.0,900.0);
        glVertex2f(2000.0,900.0);  //bridge top 4
        glVertex2f(2000.0,1400.0);
        glVertex2f(1300.0,1400.0);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(1.0,1.0,0.0);
        glVertex2f(1300.0,1370.0);
        glVertex2f(2000.0,1370.0);
        glVertex2f(2000.0,1375.0);   //yellow strip3
        glVertex2f(1300.0,1375.0);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(0.65,0.45,0.45);
        glVertex2f(1300.0,1150.0);
        glVertex2f(2000.0,1150.0);
        glVertex2f(2000.0,1180.0);   //divider
        glVertex2f(1300.0,1180.0);
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(1400.0,1280.0);
        glVertex2f(1500.0,1280.0);
        glVertex2f(1500.0,1290.0);  //strip3 above 3rd
        glVertex2f(1400.0,1290.0);
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();

glBegin(GL_POLYGON);
        glColor3f(1.0,1.0,1.0);
        glVertex2f(1600.0,1280.0);
        glVertex2f(1700.0,1280.0);
        glVertex2f(1700.0,1290.0);  //strip4 above 4th
        glVertex2f(1600.0,1290.0);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(1.0,1.0,1.0);
        glVertex2f(1800.0,1030.0);
        glVertex2f(1900.0,1030.0);
        glVertex2f(1900.0,1040.0);  //strip5
        glVertex2f(1800.0,1040.0);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(1.0,1.0,1.0);
        glVertex2f(1800.0,1280.0);
        glVertex2f(1900.0,1280.0);
        glVertex2f(1900.0,1290.0);  //strip5 above 5th
        glVertex2f(1800.0,1290.0);
glEnd();

glBegin(GL_LINES);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(1725.0,1550.0);
        glVertex2f(900.0+k,1400.0+n);   //right pole thread
        glVertex2f(1745.0,1550.0);
        glVertex2f(900.0+k,1400.0+n);
        glEnd();
```

```
	glBegin(GL_POLYGON);
		glColor3f(0.25,0.25,0.25);
		glVertex2f(200.0,800.0);   //6 point polygon 1
		glVertex2f(200.0,700.0);
		glVertex2f(300.0,700.0);
		glVertex2f(300.0,800.0);
		glVertex2f(350.0,880.0);
		glVertex2f(150.0,880.0);
		glEnd();

	glBegin(GL_POLYGON);
		glColor3f(0.0,0.0,0.0);
		glVertex2f(0.0,880.0);
		glVertex2f(500.0,880.0);   //base3
		glVertex2f(500.0,900.0);
		glVertex2f(0.0,900.0);
	glEnd();

	glBegin(GL_POLYGON);
		glColor3f(0.0,0.0,0.0);    //base4
		glVertex2f(1300.0,880.0);
		glVertex2f(2000.0,880.0);
		glVertex2f(2000.0,900.0);
		glVertex2f(1300.0,900.0);
	glEnd();

	glBegin(GL_POLYGON);
		glColor3f(0.25,0.25,0.25);
		glVertex2f(1500.0,800.0);
		glVertex2f(1500.0,700.0);
		glVertex2f(1600.0,700.0);  //6 point polygon2
		glVertex2f(1600.0,800.0);
		glVertex2f(1650.0,880.0);
		glVertex2f(1450.0,880.0);
	glEnd();
	}

////////////////////////////////////////  Boat function  ////////////////////////////////
void boat(){
		glPushMatrix();
		glTranslatef(0,y,0);
		glPushMatrix();
	glBegin(GL_POLYGON);
		glColor3f(.75,.75,.75);
		glVertex2f(900.0,700.0);
		glVertex2f(800.0,620.0);
		glVertex2f(750.0,500.0);
		glVertex2f(750.0,200.0);          //ship
		glVertex2f(900.0,50.0);
		glVertex2f(1050.0,200.0);
```

```
        glVertex2f(1050.0,500.0);
        glVertex2f(1000.0,620.0);
glEnd();

glBegin(GL_POLYGON);
        glColor3f(1.0,0.0,0.0);          // 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);
        glVertex2f(905,570);
        glVertex2f(905,722);
        glColor3f(0.0,0.0,0.0);          //flag pole
        glVertex2f(910,722);
        glVertex2f(910,570);
        glEnd();

glBegin(GL_POLYGON);
        glVertex2f(910,640);
        glVertex2f(980,720);
        glColor3f(0.5,0.5,0.5);          //flag triangle
        glVertex2f(910,720);
        glVertex2f(980,640);
glEnd();

glBegin(GL_POLYGON);
        glVertex2f(800,300);
        glVertex2f(800,406);
        glColor3f(0.8,0.4,0.7);          //pillar
        glVertex2f(850,406);
        glVertex2f(850,300);
        glEnd();

glBegin(GL_POLYGON);
        glColor3f(0,0,0);
        glVertex2f(901.0,0.0);           //ship back  2
        glVertex2f(1050.0,200.0);
        glVertex2f(1051.0,200.0);
        glVertex2f(901.0,50.0);
glEnd();

glBegin(GL_LINES);
        glColor3f(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);
        glVertex2f(750.0,500.0);
        glVertex2f(770.0,500.0);
        glVertex2f(770.0,200.0);
        glVertex2f(770.0,200.0);
        glVertex2f(750.0,200.0);
        glVertex2f(770.0,200.0);
        glVertex2f(900.0,70.0);
        glVertex2f(900.0,70.0);
        glVertex2f(900.0,50.0);
        glVertex2f(900.0,70.0);
        glVertex2f(1030.0,200.0);
        glVertex2f(1030.0,200.0);
        glVertex2f(1050.0,200.0);
        glVertex2f(1030.0,200.0);
        glVertex2f(1030.0,500.0);
        glVertex2f(1030.0,500.0);
        glVertex2f(1050.0,500.0);
        glVertex2f(1030.0,500.0);
        glVertex2f(980.0,620.0);
        glVertex2f(980.0,620.0);
        glVertex2f(1000.0,620.0);
        glVertex2f(980.0,620.0);
        glVertex2f(900.0,700.0);
        glVertex2f(770.0,350.0);
        glVertex2f(750.0,350.0);
        glVertex2f(770.0,450.0);
        glVertex2f(750.0,450.0);
        glVertex2f(770.0,250.0);
        glVertex2f(750.0,250.0);
        glVertex2f(1030.0,250.0);
        glVertex2f(1050.0,250.0);
        glVertex2f(1030.0,350.0);
        glVertex2f(1050.0,350.0);
        glVertex2f(1030.0,450.0);
        glVertex2f(1050.0,450.0);
        glVertex2f(840.0,130.0);
        glVertex2f(820.0,110.0);
        glVertex2f(975.0,110);
        glVertex2f(955.0,125.0);
glEnd();

        glBegin(GL_POLYGON);
                glColor3f(0.4,0.42,0.5);
                glVertex2f(800.0,200.0); //ship compartment 1
                glVertex2f(1000.0,200.0);
                glVertex2f(1000.0,500.0);
                glVertex2f(800.0,500.0);
        glEnd();
```

```
        glBegin(GL_POLYGON);
                glColor3f(0.6,0.65,0.7);
                glVertex2f(830.0,250.0); //ship compartment 2
                glVertex2f(970.0,250.0);
                glVertex2f(970.0,450.0);
                glVertex2f(830.0,450.0);
        glEnd();

glBegin(GL_POLYGON);
        glVertex2f(840,450);
        glVertex2f(840,500);
        glColor3f(0.3,0.1,0.75);            //flag triangle 1
        glVertex2f(800,500);
        glVertex2f(840,450);
glEnd();

glBegin(GL_POLYGON);
        glVertex2f(838,390);
        glVertex2f(838,500);
        glColor3f(0.35,0.1,0.75);                   //pillar 1
        glVertex2f(842,500);
        glVertex2f(842,390);
        glEnd();

glBegin(GL_POLYGON);
        glVertex2f(962,450);
        glVertex2f(1002,500);
        glColor3f(0.3,0.1,0.75);            //flag triangle 2
        glVertex2f(962,500);
        glVertex2f(962,450);
glEnd();

glBegin(GL_POLYGON);
        glVertex2f(959,390);
        glVertex2f(959,500);
        glColor3f(0.35,0.1,0.75);                   //pillar 2
        glVertex2f(962,500);
        glVertex2f(962,390);
        glEnd();

glBegin(GL_POLYGON);
        glVertex2f(962,300);
        glVertex2f(1002,350);
        glColor3f(0.3,0.1,0.75);            //flag triangle 3
        glVertex2f(962,350);
        glVertex2f(962,300);
glEnd();

glBegin(GL_POLYGON);
        glVertex2f(960,270);
        glVertex2f(960,350);
```

```
                glColor3f(0.35,0.1,0.75);                   //pillar 3
                glVertex2f(963,350);
                glVertex2f(963,270);
                glEnd();

        glBegin(GL_POLYGON);
                glVertex2f(840,300);
                glVertex2f(840,350);
                glColor3f(0.3,0.1,0.75);            //flag triangle 4
                glVertex2f(800,350);
                glVertex2f(840,300);
        glEnd();

        glBegin(GL_POLYGON);
                glVertex2f(838,270);
                glVertex2f(838,350);
                glColor3f(0.35,0.1,0.75);                   //pillar 4
                glVertex2f(841,350);
                glVertex2f(841,270);
                glEnd();

         glPopMatrix();
        glPopMatrix();
}

//////////////////////////////////////////   bus/car function   ////////////////////////////////
void bus1(){
glPushMatrix();
glTranslatef(g,0,0);
        glBegin(GL_POLYGON);            //  car top red
                glColor3f(0.9,0.15,0.15);
                glVertex2f(1800.0,1050.0);
                glVertex2f(1950.0,1050.0);
                glVertex2f(1950.0,1150.0);
                glVertex2f(1800.0,1150.0);
        glEnd();

        glBegin(GL_POLYGON);            //  car top black
                glColor3f(0.0,0.0,0.0);
                glVertex2f(1770.0,1030.0);
                glVertex2f(1800.0,1050.0);
                glVertex2f(1800.0,1150.0);
                glVertex2f(1770.0,1130.0);
        glEnd();

        glBegin(GL_POLYGON);            //  car side black
                glColor3f(0.0,0.0,0.0);
                glVertex2f(1770.0,1030.0);
                glVertex2f(1930.0,1030.0);
                glVertex2f(1950.0,1050.0);
                glVertex2f(1800.0,1050.0);
```

```
        glEnd();

          glBegin(GL_LINES);
                glColor3f(0.0,0.0,0.0);
                glVertex2f(1820.0,1080.0);
                glVertex2f(1920.0,1080.0);
                glVertex2f(1920.0,1080.0);
                glVertex2f(1920.0,1110.0);
                glVertex2f(1920.0,1110.0);          //carrier
                glVertex2f(1820.0,1110.0);
                glVertex2f(1820.0,1110.0);
                glVertex2f(1820.0,1080.0);
                glVertex2f(1840.0,1080.0);
                glVertex2f(1840.0,1110.0);
                glVertex2f(1860.0,1080.0);
                glVertex2f(1860.0,1110.0);
                glVertex2f(1880.0,1080.0);
                glVertex2f(1880.0,1110.0);
                glVertex2f(1900.0,1080.0);
                glVertex2f(1900.0,1110.0);
                glEnd();

        glBegin(GL_LINES);
                glColor3f(1.0,0.0,0.0);
                glVertex2f(1780.0,1035.0);      //head lamp
                glVertex2f(1780.0,1045.0);
        glEnd();

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

        glBegin(GL_LINES);
                glColor3f(1.0,0.0,0.0);
                glVertex2f(1790.0,1055.0);      //horn grill
                glVertex2f(1790.0,1125.0);
                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();
}
//////////////////////////////////////////////////2nd
car//////////////////////////////////////////
void bus2(){
```

```
glPushMatrix();
glTranslatef(-g,0,0);
        glBegin(GL_POLYGON);            //  cartop red
                glColor3f(1.0,0.5,0.0);
                glVertex2f(180.0,1250.0);
                glVertex2f(300.0,1250.0);
                glVertex2f(300.0,1340.0);
                glVertex2f(180.0,1340.0);
        glEnd();

        glBegin(GL_POLYGON);            //  car black
                glColor3f(0.0,0.0,0.0);
                glVertex2f(300.0,1250.0);
                glVertex2f(335.0,1230.0);
                glVertex2f(335.0,1330.0);
                glVertex2f(300.0,1350.0);
        glEnd();

        glBegin(GL_POLYGON);            //  car side black
                glColor3f(0.0,0.0,0.0);
                glVertex2f(180.0,1250.0);
                glVertex2f(310.0,1250.0);
                glVertex2f(330.0,1230.0);
                glVertex2f(200.0,1230.0);
        glEnd();

        glBegin(GL_LINES);
                glColor3f(0.0,0.0,0.0);
                glVertex2f(200.0,1280.0);
                glVertex2f(300.0,1280.0);
                glVertex2f(300.0,1280.0);
                glVertex2f(300.0,1310.0);
                glVertex2f(300.0,1310.0);       //carrier
                glVertex2f(200.0,1310.0);
                glVertex2f(200.0,1310.0);
                glVertex2f(200.0,1280.0);
                glVertex2f(200.0,1280.0);
                glVertex2f(200.0,1310.0);
                glVertex2f(240.0,1280.0);
                glVertex2f(240.0,1310.0);
                glVertex2f(260.0,1280.0);
                glVertex2f(260.0,1310.0);
                glVertex2f(300.0,1280.0);
                glVertex2f(300.0,1310.0);
        glEnd();

        glBegin(GL_LINES);
                glColor3f(1.0,0.5,0.0);
                glVertex2f(325.0,1245.0);     //head lamp
                glVertex2f(325.0,1255.0);
        glEnd();
```

```
	glBegin(GL_LINES);
		glColor3f(1.0,0.5,0.0);
		glVertex2f(325.0,1325.0);	//head lamp
		glVertex2f(325.0,1335.0);
	glEnd();

	glBegin(GL_LINES);
		glColor3f(1.0,0.5,0.0);
		glVertex2f(315.0,1255.0);	//horn grill
		glVertex2f(315.0,1325.0);
		glEnd();

	glBegin(GL_LINES);
		glColor3f(1.0,0.5,0.0);
		glVertex2f(200.0,1240.0);	//side window
		glVertex2f(300.0,1240.0);
		glEnd();

	glPopMatrix();
}

//////////////////////////////////////// bus3 ////////////////////////////////
void bus3(){
glPushMatrix();
glTranslatef(g,0,0);
	glBegin(GL_POLYGON);		// car top red
		glColor3f(0.75,0.75,0.95);
		glVertex2f(1500.0,1050.0);
		glVertex2f(1600.0,1050.0);
		glVertex2f(1600.0,1150.0);
		glVertex2f(1500.0,1150.0);
	glEnd();

	glBegin(GL_POLYGON);		// car top black
		glColor3f(0.0,0.1,0.0);
		glVertex2f(1450.0,1030.0);
		glVertex2f(1500.0,1050.0);
		glVertex2f(1500.0,1150.0);
		glVertex2f(1450.0,1130.0);
	glEnd();

	glBegin(GL_POLYGON);		// car side black
		glColor3f(0.0,0.0,0.0);
		glVertex2f(1450.0,1030.0);
		glVertex2f(1590.0,1030.0);
		glVertex2f(1600.0,1050.0);
		glVertex2f(1500.0,1050.0);
	glEnd();

	glBegin(GL_LINES);
```

```
                glColor3f(0.75,0.75,0.95);
                glVertex2f(1470.0,1035.0);      //head lamp
                glVertex2f(1470.0,1045.0);
        glEnd();
}

////////////////////////////////////////////// Pole Function ////////////////////////////
void poles(){
        glBegin(GL_POLYGON);            //  left pole behind
                glColor3f(0.0,0.0,0.0);
                glVertex2f(30.0,1400.0);
                glVertex2f(50.0,1400.0);
                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(1725.0,1400.0);
                glVertex2f(1745.0,1400.0);
                glVertex2f(1745.0,1550.0);
                glVertex2f(1725.0,1550.0);
        glEnd();

        glBegin(GL_LINES);
                glColor3f(0.0,0.0,0.0);
                glVertex2f(1750.0,1400.0);
                glVertex2f(900.0+k,900.0+n);   //right pole thread front
                glVertex2f(1770.0,1400.0);
                glVertex2f(900.0+k,880.0+n);
                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_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);

void myinit(){
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,1.0,1.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,2000.0,0.0,1600.0);
}

//////////////////////////////// Menu function //////////////////////

void main_menu(int ch){
        switch(ch){
        case 1:glutIdleFunc(NULL);break;

        case 2:glutIdleFunc(animate);break;

        case 3:exit(0);
        }
}


void strokeString(float x, float y, float sx, float sy, char *string, int width){
  char *c;
  glLineWidth(width);
  glPushMatrix();
  glTranslatef(x, y, 0);
  glScalef(sx, sy, 0);
  for (c = string; *c != '\0'; c++)
    glutStrokeCharacter(GLUT_STROKE_ROMAN, *c);
  glPopMatrix();
}

void first(){
        glClear(GL_COLOR_BUFFER_BIT);

  glColor3f(0.15, 0, 0.55);
    strokeString(450, 1400, 0.5, 0.5, "BMS INSTITUTE OF TECHNOLOGY", 5);

  glColor3f(0.0, 0.0, 0.0);
        strokeString(370, 1270, 0.5, 0.5, "DEPARTMENT OF COMPUTER SCIENCE", 5);
        strokeString(700, 1150, 0.5, 0.5, "AND ENGINEERING", 5);
    glColor3f(0.5, 0.0, 0.1);
        strokeString(320, 980, 0.5, 0.5, "COMPUTER GRAPHICS MINI PROJECT ON", 4);

        glColor3f(0.15, 0, 0.55);
        strokeString(580, 800, 0.8, 0.8, "BASCULE BRIDGE", 6);
```

```
        glColor3f(0.5, 0, 0.1);
        strokeString(950, 630, 0.35, 0.35, "By:", 3);

        glColor3f(0,0,0);
        strokeString(720, 550, 0.35, 0.35, "S Satish - 1BY18CS132", 3);
        strokeString(720, 480, 0.35, 0.35, "S Arshad - 1BY18CS227", 3);
        strokeString(720, 420, 0.35, 0.35, "Chirag V - 1BY18CS228", 3);

   glColor3f(0.5, 0, 0.1);
      strokeString(1600, 200, 0.2, 0.2, "Press (SPACE + ENTER)", 2);

        glutSwapBuffers();

}

void initfirst(){
   glClearColor(0.5, 0.4,0.3, 0.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0, 1000, 0, 1000, -1, 1);
   glMatrixMode(GL_MODELVIEW);
}

int main(int argc,char **argv){
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(1500,1500);
        glutCreateWindow("BASCULE BRIDGE");
        initfirst();
        glutDisplayFunc(first);
        glutKeyboardFunc(key);
        //glutIdleFunc(idle);

        glutCreateMenu(main_menu);
        glutAddMenuEntry("Stop Animation",1);
        glutAddMenuEntry("Start Animation",2);
        glutAddMenuEntry("Quit",3);

        glutAttachMenu(GLUT_RIGHT_BUTTON);

        myinit();
        glClearColor (0.5,0.8,0.55,0);
        glutMainLoop();
      return 0;
}

//////////////////////////////////  The End  :D  ////////////////////////
```

THE END