ABSTRACT

‘A picture is worth a thousand words’ goes the ancient Chinese proverb. This has become a cliché in our society after the advent of inexpensive and simple techniques for producing pictures.

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.

CONTENTS

Chapter 1

INTRODUCTION

Although computer graphics is a vast field that encompasses almost any graphical aspect, we are mainly interested in the generation of images of 2 and 3-dimensional scenes. Computer imagery has applications for film special effects, simulation and training, games, medical imagery, flying logos, etc. Computer graphics relies on an internal model of the scene, that is, a mathematical representation suitable for graphical computations. One such real life scene we found interesting is the working of the Lift-over Bridge or the Bascule Bridge and we have tried to incorporate this in this project.

A Bascule bridge (also called the Draw Bridge) is a movable bridge with a counter weight that continuously balances the span or "Leaf" throughout the entire upward swing in providing clearance for water traffic.

Bascule is basically a French term for 'seesaw' and 'balance'. 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. Although the Bascule Bridge has been in use since ancient times, it was not until the 1850s that engineers developed the ability to move very long, heavy spans quickly for practical purposes. The main advantage of Lift Over bridges are:

- It lifts the bascules and allows the ships and other water way bodies under it.
- The lift over bridge would be having two bascules which seem to be a single block when the bridge is closed. Whenever it is necessary to open the bascules, the bridge would take the help of towers which would be constructed on either side of the bascules. These towers act as supporting strength for the opening and closing bascules.
- The heavy cable helps to lift the bascules and place them back down. These cables also determine the speed at which the bascules should lift up and down across the water bodies.

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.

One of the most famous examples of the bascules is the Tower Bridge, which spans the River Thames just below London Bridge. It is the most distinctive of London's bridges and its construction was a masterly engineering achievement. The building of the Tower Bridge came about because the development of cross-Thames traffic had far outstripped the capacity of the existing bridges.

The lift over bridge performs following functions:

- 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.

The Lift over bridge which performs all these functions has been implemented using OpenGL functions and contains the Menu options and the Keyboard interface. It has the following features and performs the following functions:

- OpenGL based bridge which lifts its roadway automatically whenever a ship sails towards it.
- The bridge automatically returns to its normal position when a ship sails under it.
- A vehicle (car or bus) travels over the bridge.
- Other Options includes START ANIMATION, STOP ANIMATION and EXIT.
- The project is implemented on C platform with the help of OpenGL in-built functions. Care is taken to provide an easy-to-use mouse and keyboard interface involving an icon based interaction.

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.

3.1.1 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.

3.1.2 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:

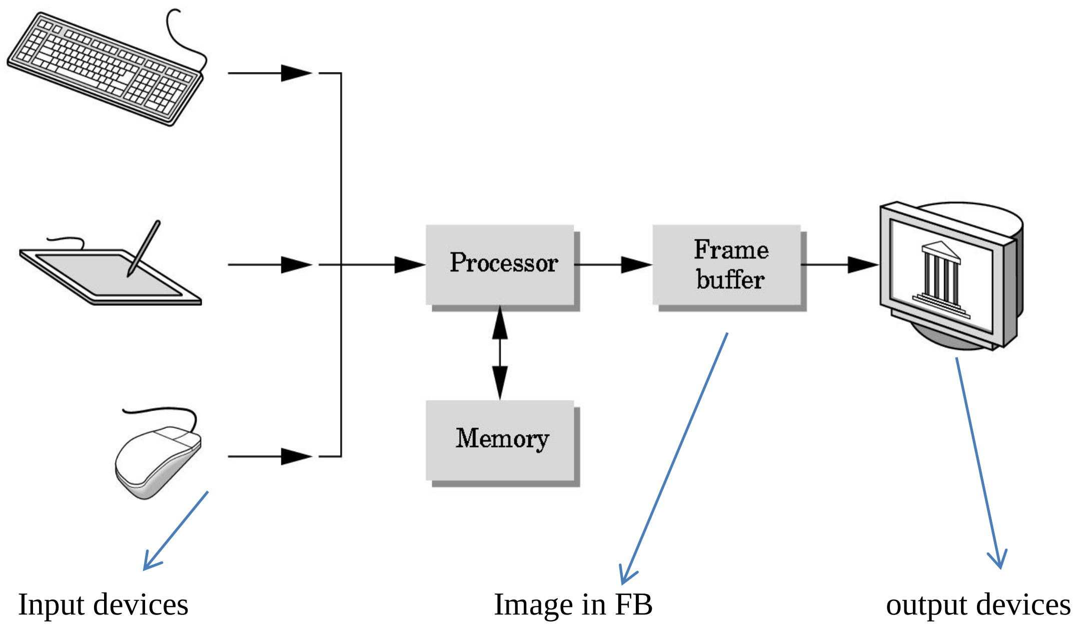


Fig 2.1: Basic Graphics System.

3.2 About OpenGL -

OpenGL is an open specification for an applications program interface for defining 2D and 3D objects. The specification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating system using any OpenGL-adhering graphics adapter.

In Computer graphics, a 3-dimensional primitive can be anything from a single point to an 'n' sided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

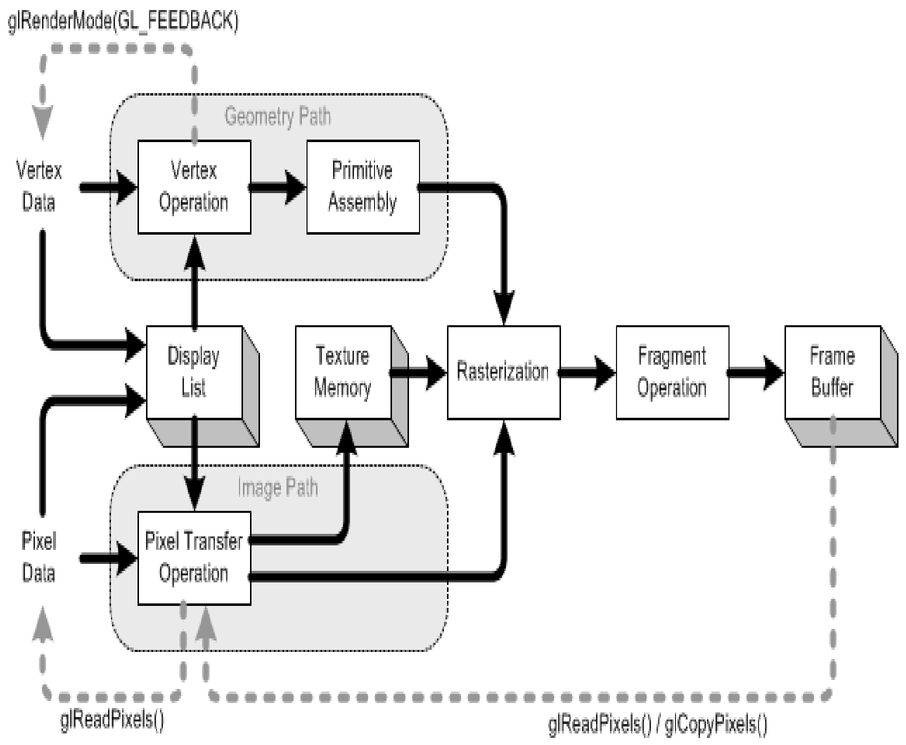


Fig 2.2: The OpenGL rendering pipeline.

3.3 Advantages of OpenGL-

- With different 3D accelerators, by presenting the programmer to hide the complexities of interfacing with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).

Chapter 3

REQUIREMENT SPECIFICATION

3.1 Hardware requirements- Pentium 3 or higher processor, 128 MB or more RAM, a standard keyboard, an Ubuntu Linux compatible mouse and a VGA monitor. If the user wants to save the created files a secondary storage medium can be used.

3.2 Software requirements- The graphics package has been designed for Ubuntu OS, hence the machine must have Eclipse installed preferably 6.22 or later versions with the glut library functions installed. Other graphics libraries are used.

3.3 Development platform- Ubuntu 10.04 and Eclipse

3.4 Language used in coding- C language.

3.5 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 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 of START ANIMATION, STOP ANIMATION and QUIT options.
- **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 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 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 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 are red, green, blue, yellow, cyan and magenta.

The main concept behind the usage of the above 6 colors are that they are the additive and the subtractive colors:

❖ **Additive color**:

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

❖ Subtractive color:

- Form a color by filtering white light with cyan (C), Magenta (M), and Yellow (Y) filters
- Light-material interactions
- Printing
- Negative film

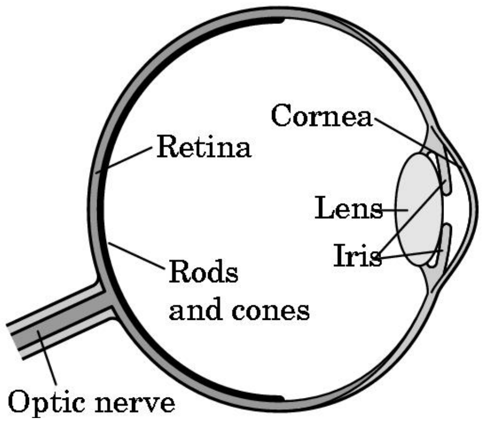


Fig: 4.1

The above figure of the eye shows the Rhodes and cones. Rhodes is responsible to detect the additive colors and the cones are in charge of detecting the subtractive colors.

• 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.

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.

4.1 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 (x_{nd}, y_{nd}) be normalized device coordinates. Then the window coordinates (x_w, y_w) are computed as follows:

$x_w = (x_{nd} + 1 \ {}^{width}/_2 + x$

$y_w = (y_{nd} + 1) \ {}^{height}/_2 + y$

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, we call the Glgetinteger with argument *GL_MAX_VIEWPORT_DIMS.*

- **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.

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. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.

If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply

- **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.

4.2 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.

➢ **int main(int argc, char**argv):**

Here we call all the function we defined previously in the program and this function creates an output window.

Chapter 6

DISCUSSIONS AND SNAPSHOTS

6.1 DISCUSSIONS:

Before we take up this topic, there we many other practical topics that we were thinking to design using the OpenGL functions. For example DOS Paint shop, Text editors which we would use in our day to day activities. Unfortunately we couldn't take up those topics since we were not able to meet the requirement at this particular stage. This made us to search for a simple and practical topic to design.

In beginning we tried to analyze topics that can be designed using simple OpenGL functions. In this way successfully we found many topics, the bascule bridge was one of that.

One of the most famous examples of the bascule type is the Tower Bridge, which spans the River Thames just below London Bridge. It is the most distinctive of London's bridges and its construction was a masterly engineering achievement. The building of the Tower Bridge came about because the development of cross-Thames traffic had far outstripped the capacity of the existing bridges. After selecting this project, the very first step that we did was to note down the possible functions that we would require implementing this scenario.

The very first object that we had to design was a river or water body that was the base for all the remaining objects. The water body was in step by step as mentioned below.

- We had to make whole display window as river so we filled the display window with blue color using the inbuilt function glColor3f (), once the window was filled with blue color our next step was to make user believe that water is flowing. This was the first real challenge that we faced while developing this project. We thought of drawing some curved lines on the window and make them move in a direction which posturizes water flow in the direction against that of curved lines. So we began to draw the curved lines. But the outcome was not as we assumed. So that left us with the option of going with the straight lines to move on the blue window.

- Our next goal was to design the bridge. We had no idea about the way in which the bridge should be designed. So we went through with many examples of designing the floor and we applied the same logic. We divided the whole bridge into four parts i.e. four strips. Two constant strips that would make the two end parts of bridge and two bascule parts that form the remaining middle position of the bridge.

These strips are designed by drawing the rectangle as shown in fig 6.1

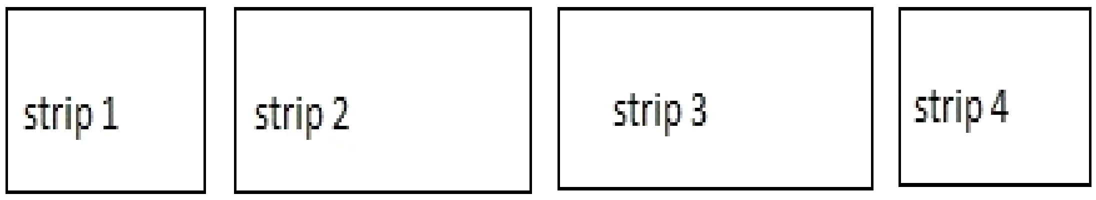


Fig 6.1

Since we designed all the four strips separate it helped us to lift the two bascules easily because we need not to lift other two strips.

- Our next part was to design the poles at the two ends of strip 2 and strip 3. After we designed the water body and the bridge, we never felt difficulty of designing these four poles. Each pole was designed by using two straight lines as shown below.

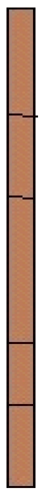
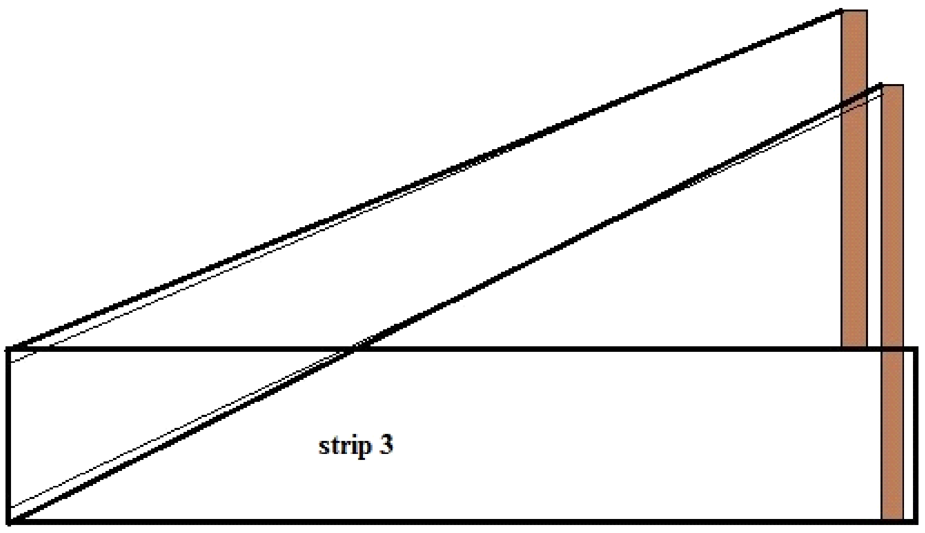


Fig 6.2

- Our next part was to connect the intersecting end of the bascules to the top of tower by drawing the straight between those two. We designed four straight lines for this purpose. We designed these four heavy cables that are used to lift up the heavy spans using the counterweight principle. The design is as shown in fig



6.3.

Fig: 6.3

•Our next duty was to design the boat. This was the toughest task for us in the whole project. First we developed a boat like image using six points and six lines that joins the points in proper manner so that it should look like a boat. The basic design that we developed as shown below :

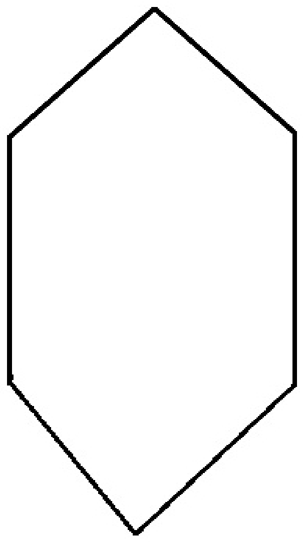


Fig: 6.4

But to implement the boat in proper way we have used 85 points at the proper place on the blue window. By joining all the points in proper manner, the boat was designed. It took more than 25 major corrections to complete this part.

- Our next step was to design a vehicle like image on the bridge, first our aim was to design more than two vehicles and make them moving in opposite direction on the bridge but it was so happened that both the vehicle like images had no space when they are crossing at the same point, that was impractical and hence we left out that idea and decided to design single vehicle. To be more precise, we decided to design a bus because of its simplicity. A bus was designed using the rectangles as shown below.

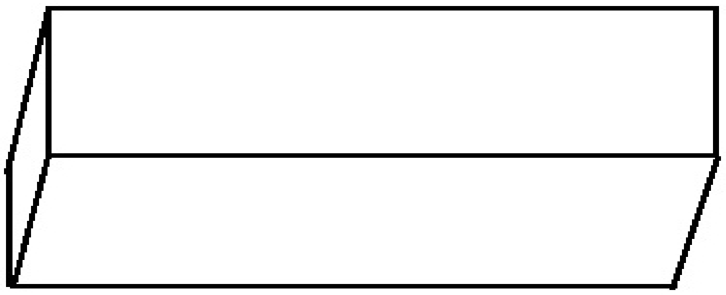


Fig: 6.5

Rest of the things i.e. using the mouse and keyboard interface for the user was provided by writing separate functions and by calling the user defined functions in proper manner.

The whole project gave some remarkable experience for us. From the beginning itself we had lots of confusions in our mind, whether the project would work as per our requirement or not. When we made an attempt to run the project as normal we encountered with many syntax errors, parameter mismatch and so on. Debugging those errors was not a difficult task for us because of the beautiful platform provided by eclipse. Once we made the corrections we execute the project at the first time the output was not exactly what we wanted so again made some changes in user defined functions to get the desired action.

6.2 SNAPSHOTS

- **Snap shot of Initial stage of the Lift over bridge :**

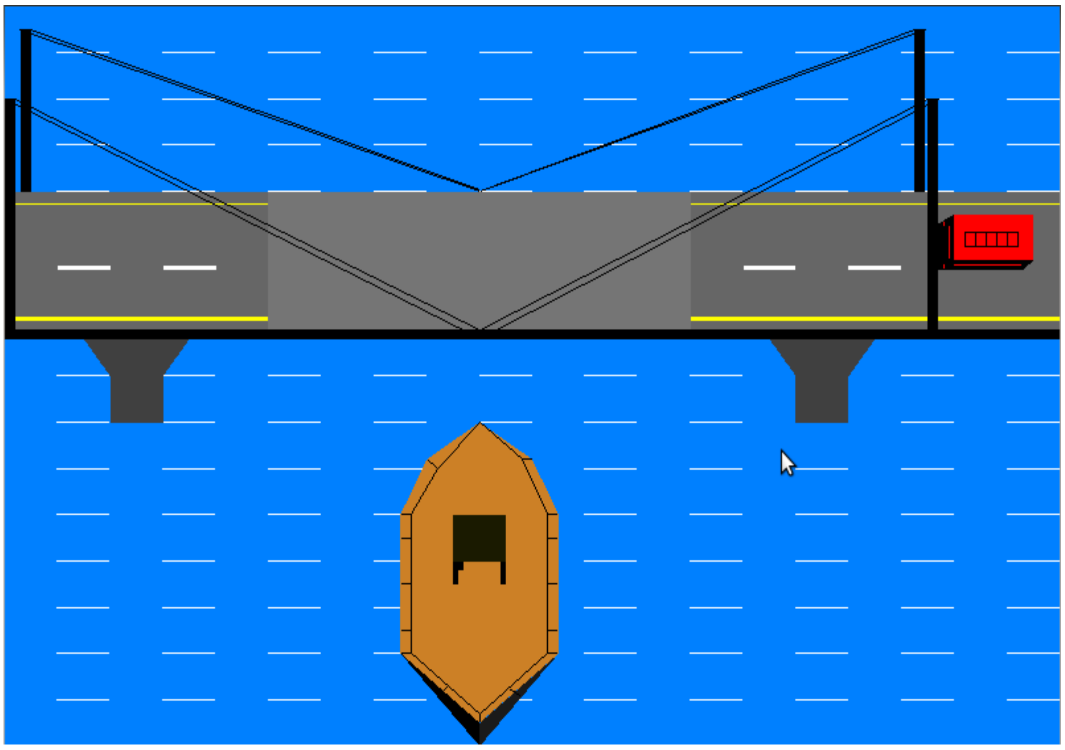


Fig: 6.2.1

- **Snap Shot Showing Main Menu (mouse interface) :**

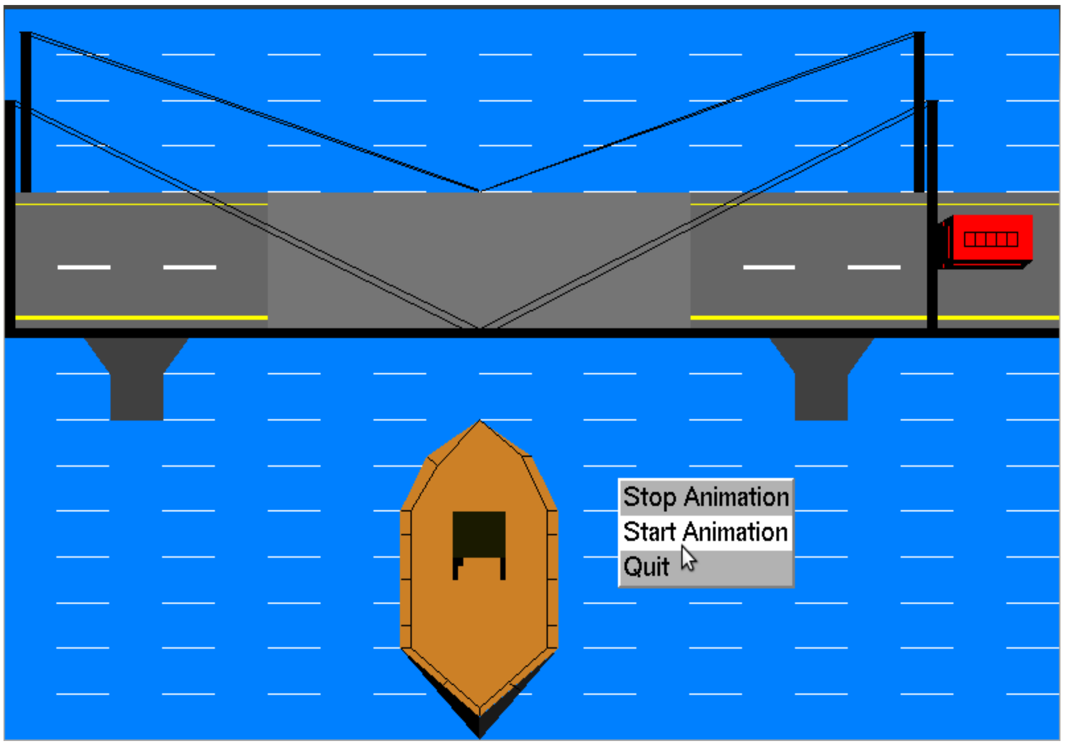


Fig: 6.2.2

- **Snapshot shows the lifting up the bascules :**

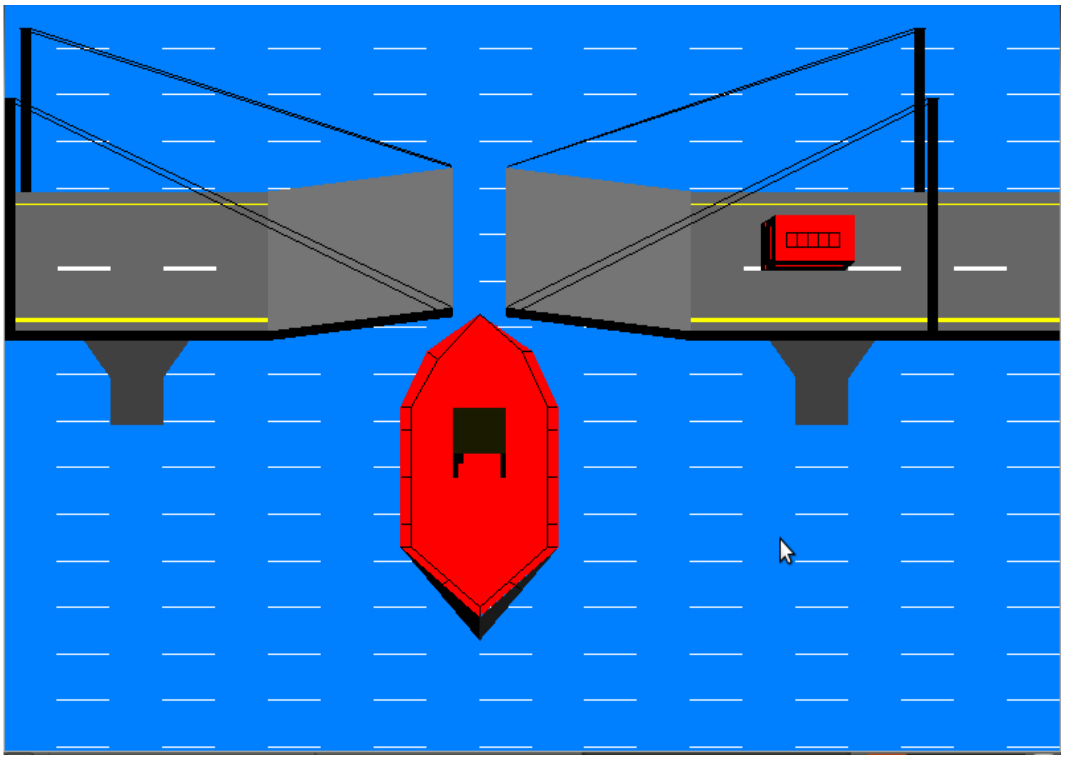


Fig: 6.2.3

- **Snap Shot Shows the changing the colors of boat:**

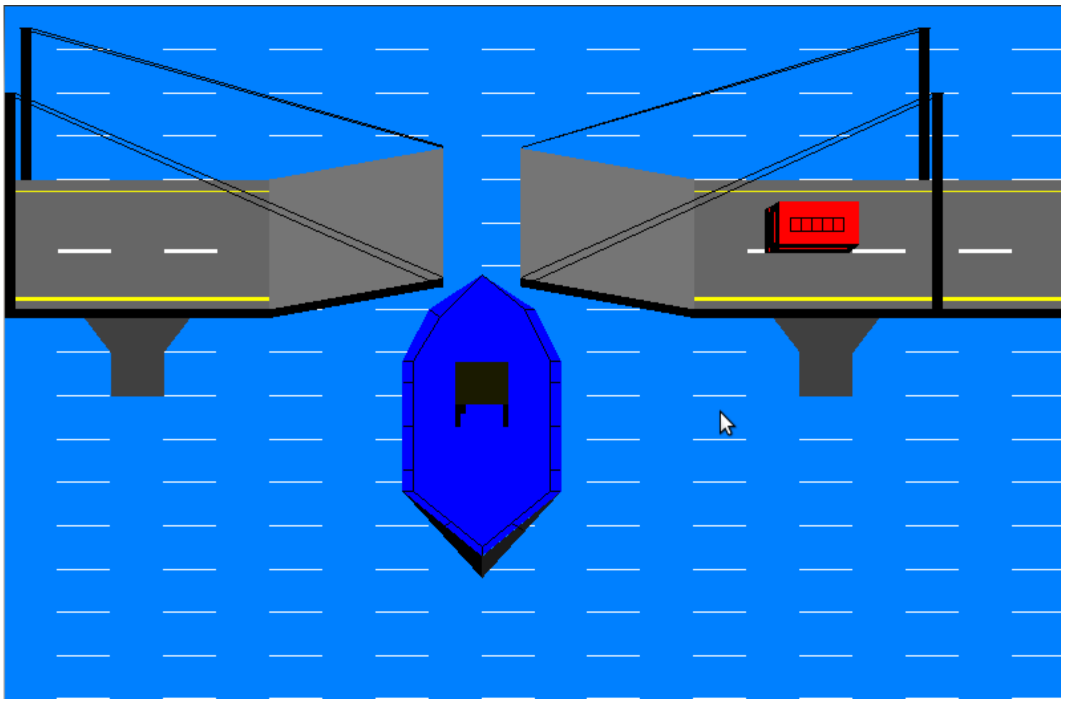


Fig: 6.2.4

- **Snap Shot Shows the Bridge moving down:**

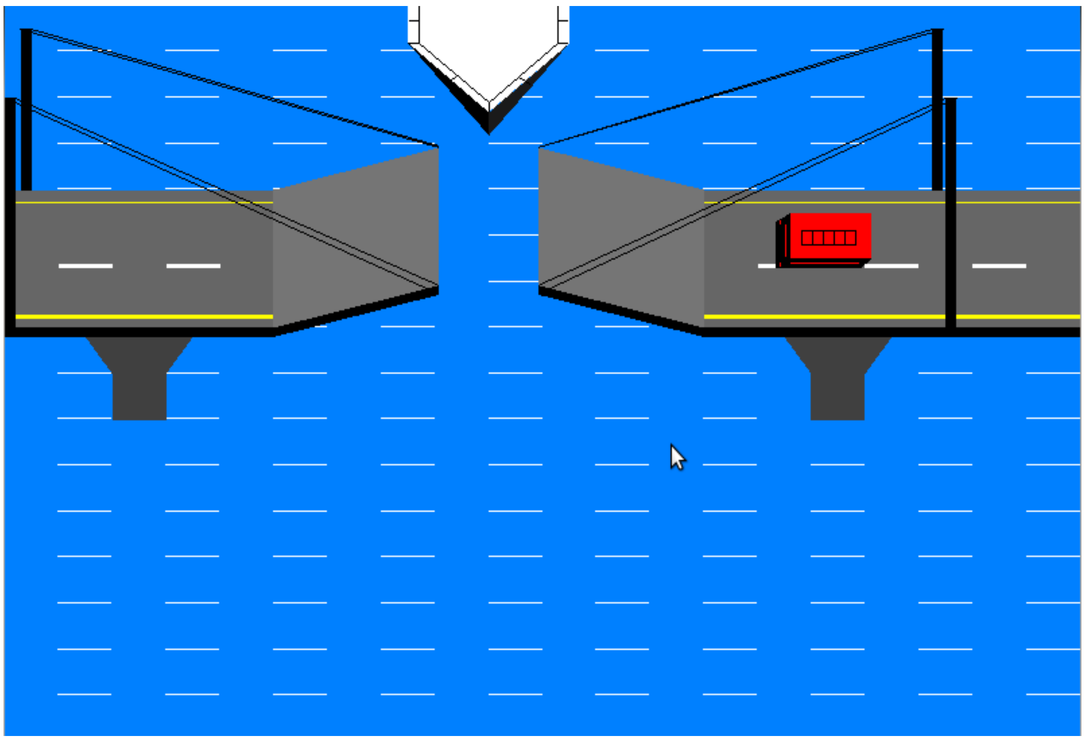


Fig: 6.2.5

- **Snap Shot Shows the Vehicle Crossing Over the Bridge:**

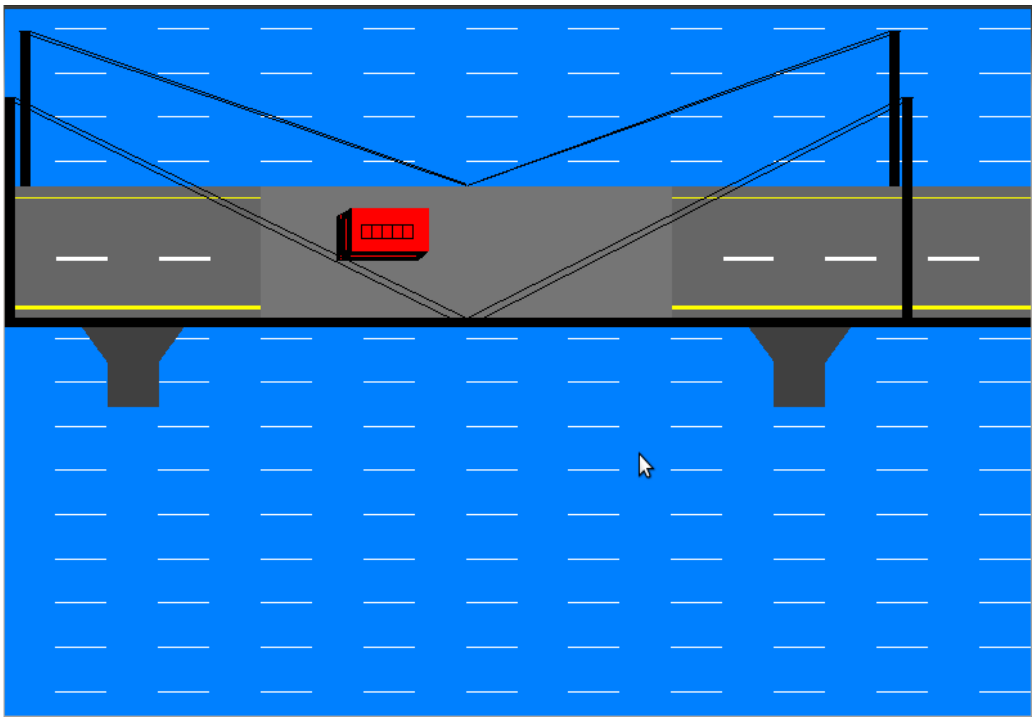


Fig: 6.2.6

Chapter 7

CONCLUSIONS AND FUTURE SCOPE

7.1 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.

7.2 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.

7.3 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.

Chapter 8

BIBLIOGRAPHY

8.1 Book References:

- Ariponnammal, S. and Natarajan, S. (1994) 'Transport Phenomena of Sm Sel-X Asx', Pramana – Journal of Physics Vol.42, No.1, pp.421-425.
- Barnard, R.W. and Kellogg, C. (1980) 'Applications of Convolution Operators to Problems in Univalent Function Theory', Michigan Mach, J., Vol.27, pp.81–94.
- Shin, K.G. and McKay, N.D. (1984) 'Open Loop Minimum Time Control of Mechanical Manipulations and its Applications', Proc.Amer.Contr.Conf., San Diego, CA, pp. 1231-1236.

8.2 Web References:

- www.opengl.org
- www.google.com
- www.sourcecode.com
- www.pearsoned.co.in
- www.wikipedia.org

Chapter 9

APPENDICES

9.1 Source Code:

```
#include<stdio.h>
#include<GL/glut.h>

 //////////////////////////////////////// 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;   // car  translate indicator
float m=.80,j=.50,o=.15;



////////////////////////////// sea function to display river   ///////////////////////////////

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();
}


///////////////////////////////////////  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(500.0,1200.0);   //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(0.0,0.0,0.0);
                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(0.46,0.46,0.46);
                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(0.0,0.0,0.0);
                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();



//              printf("i== %f\t k==%f\n",i,k);

    //===========================
    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();

    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_LINES);
                glColor3f(0.0,0.0,0.0);
                glVertex2f(1725.0,1550.0);
                glVertex2f(900.0+k,1200.0+n);    //rite pole thread
                glVertex2f(1745.0,1550.0);
                glVertex2f(900.0+k,1200.0+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(m,j,o);
                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(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(1051.0,200.0);
                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);
                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();


//  printf("g==%d\n",g);

                glBegin(GL_POLYGON);
                        glColor3f(0.10,0.10,0.);
                        glVertex2f(850.0,400.0);   //boat inside polygon
                        glVertex2f(950.0,400.0);
                        glVertex2f(950.0,500.0);
                        glVertex2f(850.0,500.0);
        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();
}



//////////////////////////////////////////    bus/car function   ////////////////////////////////
void car()
{
glPushMatrix();
```

```
glTranslatef(g,0,0);

    //glPushMatrix();
    //glTranslatef(1820.0,1030.0,0.0);
    //glutSolidSphere(1.0,200,200);
    //glPopMatrix();

    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_POLYGON);                //  car
                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
                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();
}



////////////////////////////////////////////////// Pole Function  //////////////////////////////

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_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);
   glEnd();
}



////////////////////////////// display function  //////////////////////////

void display(void)
{

                 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  sea();
  bridge();
  boat();
  car();
  poles();

                 glFlush();
                 glutSwapBuffers();
}




////////////////////////////////   function to animate bridge stripes   ////////////////////////////

void animate()
{
   q=q-.5;
   y=y+0.2;

   i+=0.2;
   if((i>=135) && (i<=439))
   {             k=k+0.1;
                 n=n+0.1;
   }
   if(i>=1200 && !(k<=0 && n<=0))
   {
                 k=k-0.1;
                 n=n-0.1;
   }

   if(k<=0)
                 g-=0.5;

   glutPostRedisplay();
}




//////////////////////////////////   myinit function ////////////////////////////////

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);
   }

}


////////////////////////////////   K/B function for changing boat color  /////////////////

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=0.0,j=0.0,o=1.0;
          glutPostRedisplay();
          break;

   case 'w':m=1.0,j=1.0,o=1.0;
                     glutPostRedisplay();
                              break;

   case 'm':m=1.0,j=.0,o=1.0;
                              glutPostRedisplay();
                              break;

   case 'c':m=.0,j=1.0,o=1.0;
                              glutPostRedisplay();
                              break;

   case 'y':m=0.0,j=1.0,o=1.0;
                              glutPostRedisplay();
                              break;

   };
}



//////////////////////////////////////////////  main function   //////////////////////////////////

int main(int argc,char **argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
```

```
    glutInitWindowSize(2000,1600);
    glutInitWindowPosition(0,0);
    glutCreateWindow("BRIDGE");

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


    glutAddMenuEntry("Quit",3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);


    glutKeyboardFunc(keyboard);
    glutKeyboardFunc(keyboard);
    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");
    printf("press 'c' to change the ship color to cyan\n");
    printf("press 'm' to change the ship color to magenta\n");
    printf("press 'w' to change the ship color to white\n");
    printf("press 'y' to change the ship color to light grey\n");
    glutDisplayFunc(display);
    myinit();
    glClearColor (1.0, 1.0, 0.0, 1.0);
    glutMainLoop();
        return 0;

}




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

9.2 Appendix (Figures):