Chapter 1

INTRODUCTION

1.1 INTRODUCTION TO COMPUTER GRAPHICS

Computer Graphics is concerned with all aspects of producing pictures or images using a computer.

Applications of Computer Graphics:

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

The Graphics Architecture:

Graphics Architecture can be made up of seven components:

1. Display processors
2. Pipeline architectures
3. The graphics pipeline
4. Vertex processing
5. Clipping and primitive assembly
6. Rasterization
7. Fragment processing

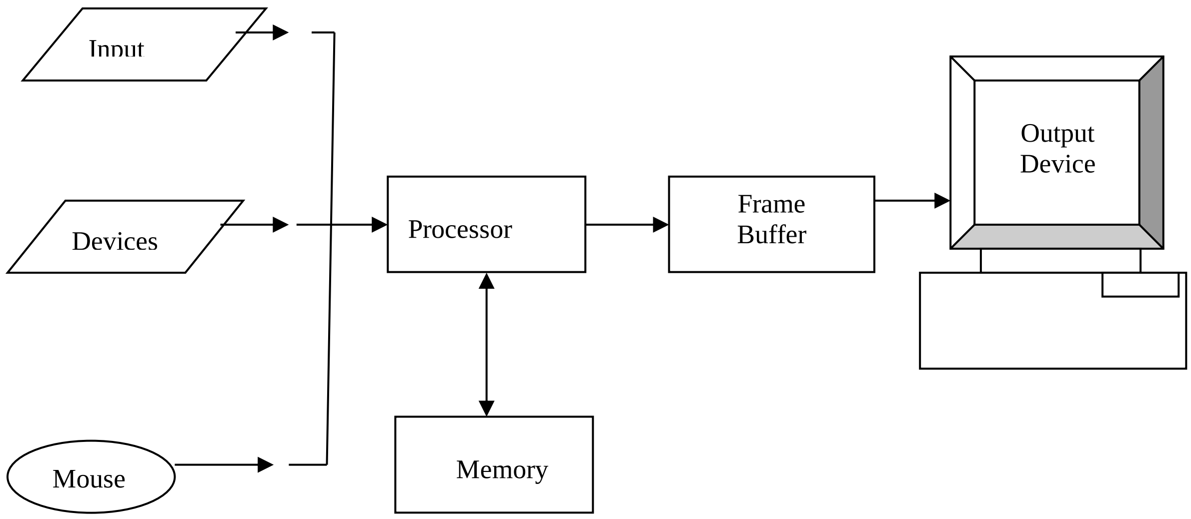


Figure 1.1: Components of Graphics Architecture and their working

1.2 ABOUT OPENGL

OpenGL is software used to implement computer graphics. The structure of OpenGL is similar to that of most modern APIs including Java 3D and DirectX. OpenGL is easy to learn, compared with other.

APIs are nevertheless powerful. It supports the simple 2D and 3D programs. It also supports the advanced rendering techniques. OpenGL API explains following 3 components

1. Graphics functions
2. Graphics pipeline and state machines
3. The OpenGL interfaces

There are so many polygon types in OpenGL like triangles, quadrilaterals, strips and fans. There are 2 control functions, which will explain OpenGL through,

1. Interaction with window system
2. Aspect ratio and view ports

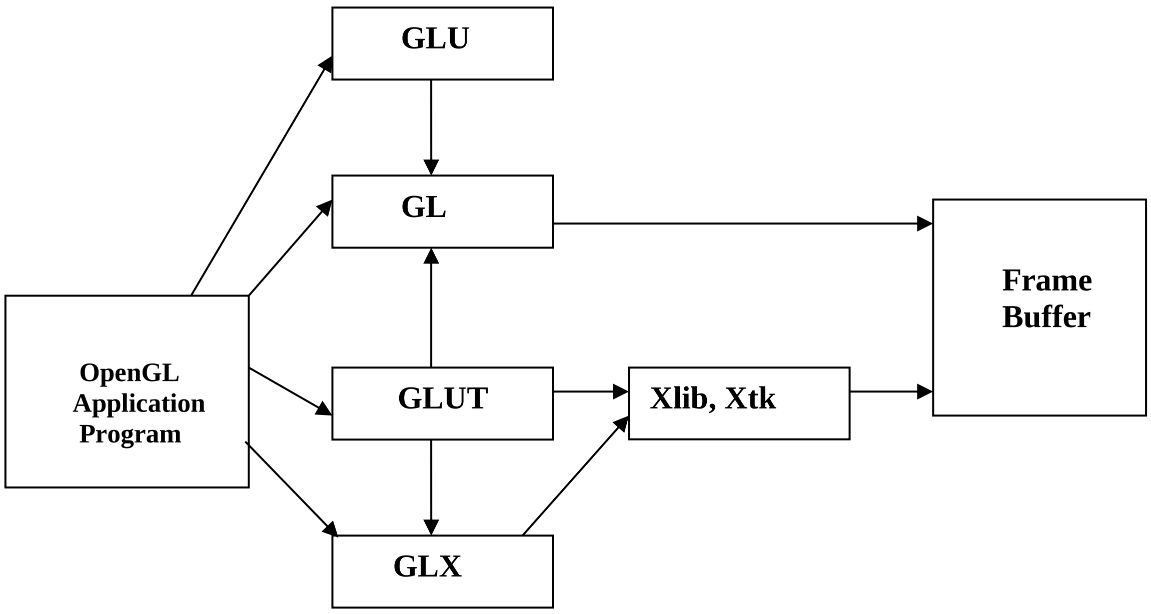


Figure 1.2: OpenGL Library organization

OpenGL (**Open G**raphics **L**ibrary) is a cross-language, multi-platform API for rendering 2D and 3D computer graphics. The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992[4] and is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games. OpenGL is managed by the non-profit technology consortium Khronos Group.

OpenGL provides the programmer with an interface to graphics hardware. It is a powerful, low-level rendering and modeling software library, available on all major platforms, games to modeling in CAD.

OpenGL intentionally provides only low-level rendering routines, allowing the programmer a great deal of control and flexibility. The provided routines can easily be used to build high-level rendering and modeling libraries, and in fact, the OpenGL Utility Library (GLU), which is included in most OpenGL distributions, does exactly that. Note also that OpenGL is just a graphics library; unlike DirectX, it does not include support for sound, input, networking, or anything else not directly related to graphics. OpenGL is also platform-independent. The specification says nothing on the subject of obtaining, and managing, an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

1.2.1 OpenGL Architecture:

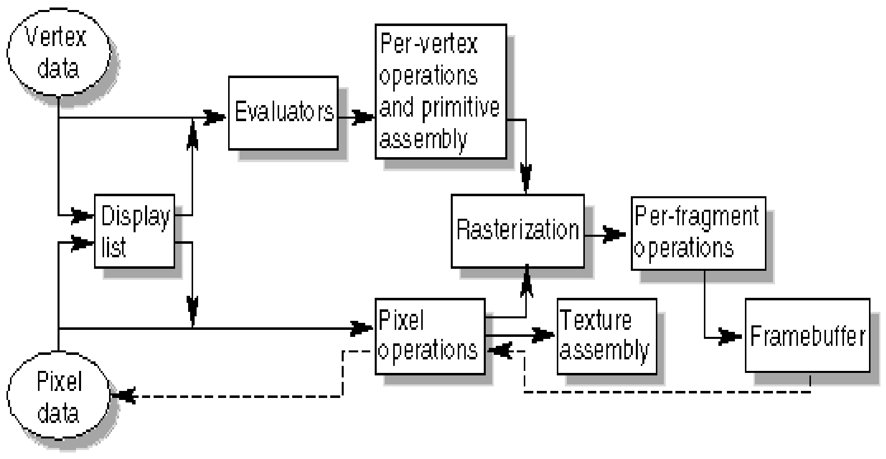


Figure 1.3: An illustrative example showing OPENGL rendering pipeline.

Evaluators

All geometric primitives are eventually described by vertices. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.

In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (z coordinate) operations are applied. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the Rasterization step.

Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the framebuffer.

Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them.

Rasterization

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

The OpenGL Utility Library:

The OpenGL Utility Library, or GLU, supplements OpenGL by providing higher-level functions. GLU offers features that range from simple wrappers around OpenGL functions to complex components supporting advanced rendering techniques. Its features include:

- 2D image scaling
- Rendering 3D objects including spheres, cylinders, and disks
- Automatic bitmap generation from a single image
- Support for curves surfaces through NURBS
- Support for tessellation of non-convex polygons
- Special-purpose transformations and matrices

What Is GLUT?

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form of windowing, menus, or input. That's where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that we can easily move GLUT-based applications from, for example, Windows to UNIX

Window Management

Five routines perform tasks necessary to initialize a window.

- **glutInit**(int *argc, char **argv) initializes GLUT and processes any command line arguments . **glutInit()** should be called before any other GLUT routine.
- **glutInitDisplayMode**(unsigned int mode) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer.
- **glutInitWindowPosition**(int x, int y) specifies the screen location for the upper-left corner of your window.
- **glutInitWindowSize**(int width, int size) specifies the size, in pixels, of your window.
- int**glutCreateWindow**(char *string) creates a window, returning a unique identifier for the new window. Until **glutMainLoop()** is called, the window is not yet displayed.

The Display Callback

glutDisplayFunc(void (*func)(void)) is an important event callback function. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, we should put all the routines that need to redraw the scene in the display callback function.

A mouse event is generated when one of the mouse buttons is pressed or released. Information returned to the program when this event occurs would include the button that generated the event, the state of the button, the position (x,y) of the cursor when the event takes place. The mouse callback function **glutMouseFunc()** must be registered in the main function.

1.3 PROBLEM DEFINITION

The program involved aims at creating a virtual scene where the graphics showed a sea with its background changing from day to night. And there are several objects that are involved.

There is also a ship moving through the scene that needs to be designed. We also need an interface definition for the controlling of the ship.

The menu also is used to provide a interactive user interface.

Chapter 2

LITERATURE SURVEY

“A picture is worth a thousand words” is a well-known saying and highlights the advantages and benefits of the visual presentation of our data. CG (Computer graphics) started with the display of data on hardcopy plotters and Cathode ray tube screen soon after the introduction of computer themselves. It includes the creation of storage and manipulation of models and images of objects. These models include physical, mathematical, engineering, architectural, and even conceptual or abstract structures and natural phenomena, and so on. Computer graphics today is largely interactive, the user controls 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 are used for user-computer interaction. A Bitmap is an one’s and zero’s representation of pixels on the screen. Bit map graphics provide easy-to-use and inexpensive graphics based applications**.**

Computer graphics can be used in many disciplines. Charting, Presentations, Drawing, Painting and design, Image processing and Scientific Visualization are some among them. Computer graphics is concerned with all aspects of producing images using a computer. It concerns with the pictorial synthesis of real or imaginary objects from their computer-based models. Video games represent a major use in the home of Computer graphics. Computer Graphics helps to create and manipulate pictures with the aid of Computers.

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

2.1 Different types of Computer Graphics:

It is further divided into two broad classes**:**

- Non-interactive Graphics
- Interactive Graphics

Non-Interactive Graphics:

In non interactive computer graphics otherwise known as passive computer graphics, the observer has no control over the image. Familiar examples of this type of computer graphics include the titles shown on TV and other forms of computer art.

Interactive Graphics:

Interactive computer graphics involves a two way communication between computer and user. Here the observer is given some control over the image by providing him with an input device for example the video game controller of the ping pong game. This helps him to signal his request to the computer. It helps to train the pilots of our airplanes.

2.2 Origin of Computer Graphics:

Years of research and development were made to achieve the goals in the field of computer graphics. In 1950, the first Computer driven display was used to generate only simple pictures. This display made us of Cathode ray tube similar to the one used in television sets. During 1950's interactive computer graphics made little progress because the computers of that period were so unsuited to interactive use. These computers were used to perform only lengthy calculations. In 1960's large computer graphics research projects were under taken at MIT, Bell Telephone Labs and General Motors. Thus the golden age of computer graphics began. In 1970's the researches began to bear fruit.

The instant appeal of computer graphics to users of all ges has helped it to spread into users of all ages has helped it to spread into many applications throughout the world.

2.3 OpenGL History

OpenGL was originally developed by Silicon Graphics, Inc. (SGI) as a multipurpose, Platform-independent graphics API. Since 1992, the development of OpenGL has been overseen by the OpenGL Architecture Review Board (ARB), which is made up of major graphics vendors and other industry leaders, currently consisting of ATI, Compaq, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, nVidia, Microsoft, and Silicon Graphics. The role of the ARB is to establish and maintain the OpenGL specification, which dictates which features must be included when one is developing an OpenGL distribution.

Because OpenGL is designed to be used with high-end graphics workstations, it has, until recently, included the power to take full advantage of consumer-level graphics hardware. Furious competition over the last couple of years, however, has brought features once available only on graphics workstations to the consumer level; as a result, there are more and more video cards of which OpenGL can't take full advantage. Eventually, these extensions may become official additions to the OpenGL standard. OpenGL 1.2 was the first version to contain support for features specifically requested by game developers (such as multi-texturing), and it is likely that future releases will be influenced by gaming as well.

Chapter 3

REQUIREMENTS & SPECIFICATION

The Requirements can be broken down into 2 major categories namely Hardware and Software requirements. The formal specification the minimal hardware facilities expected in a system in which the project has to run. The latter specifies the essential software needed to build and run the project.

3.1 HARDWARE REQUIREMENTS

There are no rigorous restrictions on the machine configurations. Since OPENGL software is portable, that is, it is platform independent or architecturally neutral, it can run on any machine with any configurations. The standard output device is assumed to be a **Color Monitor.** It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The hardware requirements are minimal and software can run with minimal requirements.

The basic requirements are as enlisted below**:**

1. Processor**:** Intel Pentium processor or a processor with higher specification.
2. Processor speed**:** 500MHz or above.
3. RAM**:**64MB or above.
4. Storage space:2MB or above.
5. Monitor resolution: A color monitor with minimum resolution of 640*480.

3.2 SOFTWARE REQUIREMENTS

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of eclipse compiler to be installed and functional in Ubuntu. Though its implemented in OpenGL, it is very much performed and independent with the restriction, that there is support for the execution of C and C++ files. Text Modes is recommended**.**

1. An MS-DOS based Operating system like Windows 95/98, Windows 2000 or Windows XP is the platform required to develop the 2D and 3D simulation.

2. A Visual C/C++ compiler like Microsoft Visual Studio 2008 is required for compiling the source code. Microsoft Office for documentation and presentation for the project.

3. A C/C++ (integrated with OpenGL) compiler like eclipse is required for compiling the source code to make the executable file which can then be directly executed.

4. A Software interface for graphics hardware with built in graphics libraries like **GL, GLU, glut and glut32**, and header files like **glut.h** are required.

Developed Platform

Windows

Language Used In Coding

C-language

Chapter 4

SOFTWARE DESIGN

4.1 SYSTEM DESIGN

Existing System

Existing system for a graphics is the **ECLIPSE**. This system supports 2D and 3D graphics. The graphics being designed should be easy to use and understand. It should provide various options such as free hand drawing, line drawing, polygon drawing, translation, rotation, scaling and clipping etc.

Proposed System

OpenGL Software was proposed in eclipse with relevant and required libraries, header files, and required primitives in it. It is a software which provides a graphical interface. It is an interface between application program and graphics hardware. The advantages are:

1. OpenGL is designed as a streamlined.
2. It is hardware independent interface i.e. it can be implemented on many different hardware platforms.
3. With OpenGL we can draw a small set of geometric primitives such as points, lines, arrays and polygons etc.
4. It provides double buffering which is vital in providing transformations.
5. It is event driven software.
6. It provides call back functions.

4.2 DETAILED DESIGN

To implement the day/night sea view we use the drawing functions. For every object in the scene a function is used to create them.

The design is simple using the key function to change the day/night view. We design two scenes one day and night and switch between them.

There is also a need to design a ship that moves through the sea at day and at night.

Chapter 5

IMPLEMENTATION

To implement day/night scene in graphics, we use routines to access features provided by the graphics hardware. The following functions were used to implement binary search algorithm in OpenGL.

5.1 FUNCTIONS USED:

5.1.1 Built-in functions:

glutInit()	Initializes GLUT. The arguments from main are passed in and can be used by the application.
glutInitDisplayMode()	Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB,GLUT_INDEX) and buffering (GLUT_SINGLE,GLUT_DOUBLE).
glutInitWindowSize()	Specifies the initial height and width of the window in pixels.
glutCreateWindow()	Creates a window on the display. The string can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.
glutDisplayFunc()	Registers the display function that is executed when the window needs to be redrawn.
glVertex()	It has many types glVertex3fv(array)- accepts an array and draws a primitive with the points in the array. glVertex2f(2 float values)- accepts 2 float values that specify the coordinates of the point. glVertex2d(2 double values) - accepts 2 double values that specify the coordinates of the point. glVertex2i(2 integer values) - accepts 2 integer values that specify the coordinates of the point. 3fv- vector(array) of 3 float values.

glutMainLoop()	Cause the program to enter an event-processing loop. It should be the last statement in main.

glMatrixMode()	Specifies which matrix will be affected by subsequent transformations. Mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.
glLoadIdentity()	Sets the current transformation matrix to an identity matrix
glOrtho()	Defines an orthographic viewing volume with all parameters measured from the center of the projection plane.
glBegin(primitive)	Begins drawing the primitive. Primitives maybe GL_POLYGON, GL_LINE_LOOP, GL_LINES, etc. GL_POLYGON- Draws a polygon with vertices as specified between glBegin() and glEnd(), that is filled with some color. GL_LINE_LOOP- Draws a polygon with vertices as specified between glBegin() and glEnd() using glVertex, that is empty. GL_LINES- Draws a line between every 2 pairs of points specified between glBegin() and glEnd() GL_POINTS-Draws a point with given co-ordinates.
glEnd()	End of glBegin()-finishes drawing the primitive
glColor()	Sets the present RGB colors. Valid types are byte (b), int (i), float (f), double (d), unsigned byte (ub), unsigned short (us) and unsigned int (ui). The maximum and minimum values of the floating point types are 1.0 and 0.0 respectively
glRect()	Supports efficient specification of rectangles as two corner points.

glRasterPosition()	Specifies a raster position.

glutBitmapCharacter()	Renders the character with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT_BITMAP_TIMES_ROMAN_10. The raster position is incremented by the width of the character
glutSwapBuffers()	Swaps the front and back buffers.
glFlush()	Forces any buffered OpenGL commands to execute
glutPostRedisplay()	Requests that the display callback be executed after the current callback returns.
glClear()	Clear buffers to present values
glutKeyboardFunc()	Specifies what should be done when keyboard is used during execution of program.
glClearColor()	Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating-point numbers between 0.0 and 1.0.
glPointSize()	Specify the diameter of rasterized points

5.2 FUNCTIONS DEFINED:

DAY

The function defines the sky and ground color to day. It also calls the mountain and the sea and the other objects defined in the day perspective. The street light is switched off.

NIGHT

The function defines the sky and ground color to night. It draws the stars and the moon. It also turns on the street lights. It calls the sea and road objects in the night perspective.

DRAWSEA, DRAWSEA_2, MOUNTAIN, MOUNTAIN_2, DRAWTREE, DRAWTREE_2, DRAWBOAT, CIRCLEDRAW

This function is used to define the objects in the scenery. The functions define the objects in 2 perspectives: night and day. The day() and night() functions call these functions to define them.

INIT

To maximize efficiency, operations that need only be called once are in this procedure.

REDRAWBOAT

This function is used to loop the image of the boat to make it move. The function Produces an image where in the boat appears to be crossing the sea with a view of land around it.

KEY

This function is used to define the keyboard functions of the program.

S : Button to stop the boat.

R : Button to reset the boat

MAIN MENU

This function is used to make a selection whether to exit the program or to start the graphics with a night scene or a day scene.

MAIN

Main function is a user defined function in which we accept the given input, and the rest of the functions are called.

Chapter 6

DISCUSSIONS & SNAPSHOTS

After the compilation step, on executing the error free code the user is prompted to:

1. The first scene shows the day view of the sea. The roads and streetlights and the other objects are visible.

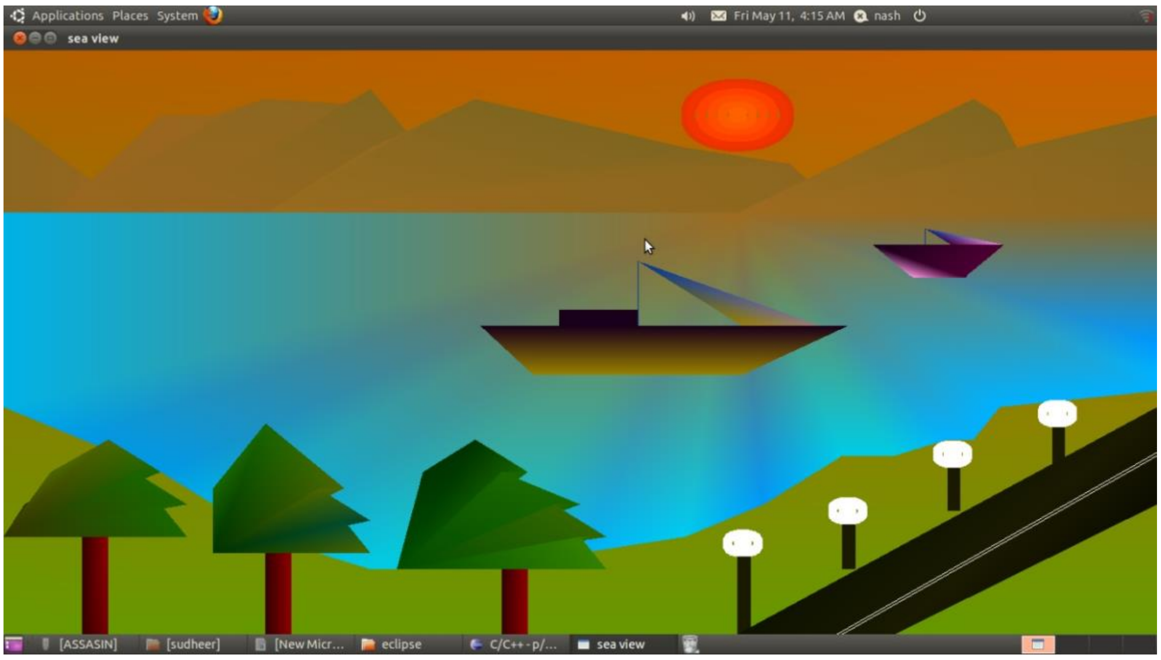


Figure 4.1 Output of Day View showing different objects in the scene

2. Night view showing the sea at night. Notice the color contrasts. The street lights appear lit with the stars.

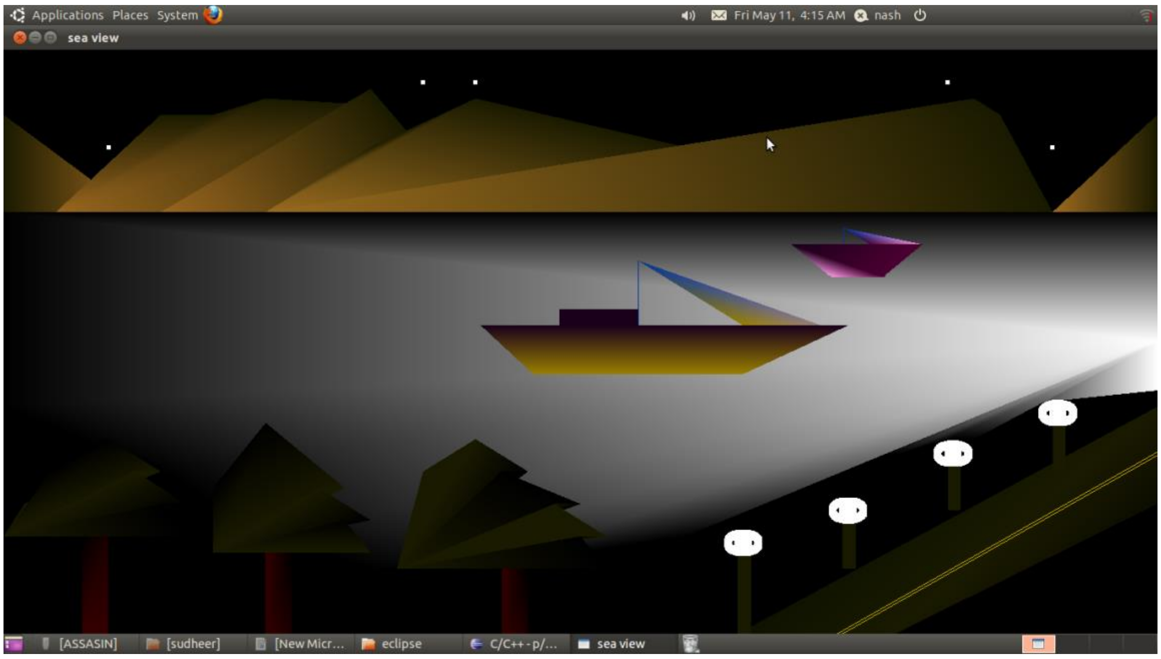
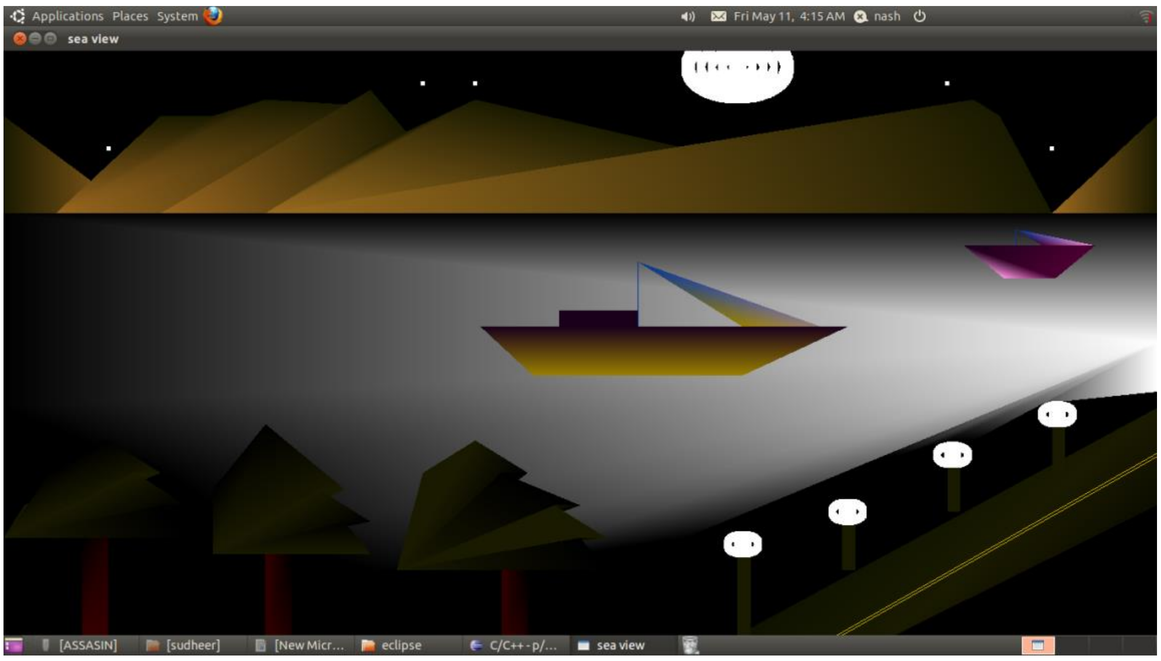


Figure 4.2 Output showing the night view of the sea

3. This image shows the smaller boat in motion along with the moon in view.



Chapter 7

CONCLUSION & FUTURE SCOPE

The program involved aims at creating a virtual scene where the graphics showed a sea with its background changing from day to night. Using in built functions provided by graphics library and integrating it with the C Language, it was possible to achieve this. After the completion of this project I got to know how we can implement a project using an open source OpenGL toolkit. I have tried my best to make this project Interactive, so that the user does not face any problem while using the simple application OpenGL graphics program.
I had a great experience in the course of doing this project, which made me to learn and understand various functions of OpenGL. With the completion of this project I have achieved a sense of happiness and I want to thank all those who helped me directly or indirectly to make this idea come true.

Pictorial visualization leads to generation of cost-effective and efficient solution to design issues.

➢ Limitations:

- The amount of detail is less.

➢ Future Enhancements:

- Making the package more user friendly.
- Can be enhanced for the efficient usage of memory.
- More options can be introduced.
- Can make the graphics more detailed.

Chapter 8

BIBLIOGRAPHY

[1] Edward Angel: Interactive Computer Graphics A Top-Down Approach Using OpenGL, Pearson Education, 5th Edition.

[2] F. S. Hill, Jr : Computer Graphics Using OpenGL Pearson Education, 3rd Edition.

[3] Foley VanDamFeiner Hughes: Computer Graphics Principles & Practice, Pearson Education, 2nd Edition in C.

[4] Donald Hearn, M. Pauline Baker : Computer Graphics 'C' version Pearson Education, 2nd Edition.

[5] http://www.opengl.org/

[6] http://www.academictutorials.com/

[7] http://www.glprogramming.com/

Chapter 9

APPENDIX

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

int glob=0;
int boat=0;
int n;
int bi;
int s;

void day();
void night();
void drawsea();
void drawsea2();
void mountain();
void mountain2();
void drawtree2();
void drawtree();
void drawboat();
void redrawboat();
void drawsmallboat();
void circle_draw(GLint h,GLint k,GLint r);
void plotpixels(GLint h,GLint k,GLint x,GLint y);
void display();


void key(unsigned char key,int x,int y);
void mainmenu(int id);

void init();
int main(int argc,char** argv);
```

```
void init()
{         glClearColor(.8,.5,0.0,1.0);

          glColor3f(1.0,0.0,0.0);
          glPointSize(5.0);
          glMatrixMode(GL_PROJECTION);
          glLoadIdentity();
          gluOrtho2D(0.0,2200.0,0.0,1800.0);

          printf("press 'x' to move boat in +ve x-direction\n");
          printf("press 'y' to move boat in -ve x-direction\n");
          printf("press 'z' to move the moon in +ve y-direction\n");
          printf("press 'w' to move the moon in -ve y-direction\n");
          printf("press 's' to stop the smallboat\n");
          printf("press 'r' to restart the smallboat\n");

          glutPostRedisplay();
}

//To draw Mountain for DAY

void mountain()
{

          glColor3f(0.60,0.40,0.12);
   glBegin(GL_TRIANGLES);
          glVertex2f(250,1300);
                    glColor3f(0.55,0.40,0.12);
          glVertex2f(0,1600);
                    glColor3f(0.55,0.40,0.12);
          glVertex2f(0,1300);
          glEnd();
          glColor3f(0.60,0.40,0.12);
          glBegin(GL_POLYGON);
          glVertex2f(100,1300);
          glVertex2f(300,1600);
          glVertex2f(400,1600);
                    glColor3f(0.55,0.40,0.12);
          glVertex2f(500,1650);
          glVertex2f(700,1630);
                    glColor3f(0.55,0.40,0.12);
          glVertex2f(900,1300);
          glEnd();
```

```
        glColor3f(0.60,0.40,0.12);
        glBegin(GL_TRIANGLES);
        glVertex2f(300,1300);
        glColor3f(0.50,0.40,0.12);
        glVertex2f(700,1680);
        glColor3f(0.55,0.40,0.12);
        glVertex2f(900,1300);
        glEnd();
        glColor3f(0.60,0.42,0.12);
        glBegin(GL_POLYGON);
        glVertex2f(500,1300);
        glVertex2f(700,1500);
        glColor3f(0.50,0.40,0.12);
        glVertex2f(900,1650);
        glVertex2f(1300,1500);
        glColor3f(0.55,0.40,0.12);
        glVertex2f(1500,1450);
        glVertex2f(1600,1300);
        glEnd();
        glColor3f(0.62,0.42,0.12);
        glBegin(GL_POLYGON);
        glVertex2f(1400,1300);
        glVertex2f(1800,1600);
        glColor3f(0.50,0.40,0.12);
        glVertex2f(1850,1650);
        glVertex2f(1900,1600);
        glColor3f(0.55,0.40,0.12);
        glVertex2f(2000,1300);
        //glEnd();glColor3f(0.62,0.42,0.13);
        glBegin(GL_TRIANGLES);
        glVertex2f(2000,1300);
        glColor3f(0.50,0.40,0.12);
        glVertex2f(2200,1600);
        glColor3f(0.55,0.40,0.12);
        glVertex2f(2200,1300);

//glEnd();



}

//To Draw Mountain for Night
```

```
void mountain2()
{

        glColor3f(0.60,0.40,0.12);
   glBegin(GL_TRIANGLES);
        glVertex2f(250,1300);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(0,1600);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(0,1300);
        glEnd();
        glColor3f(0.60,0.40,0.12);
        glBegin(GL_POLYGON);
        glVertex2f(100,1300);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(300,1600);
        glColor3f(0.10,0.10,0.0);

        glVertex2f(400,1600);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(500,1650);
        glVertex2f(700,1630);
        glColor3f(0.10,0.10,0.0);

        glVertex2f(900,1300);
        glEnd();
        glColor3f(0.60,0.40,0.12);
        glBegin(GL_TRIANGLES);
        glVertex2f(300,1300);
        glColor3f(0.10,0.10,0.0);

        glVertex2f(700,1680);
        glColor3f(0.10,0.10,0.0);

        glVertex2f(900,1300);
        glEnd();
        glColor3f(0.60,0.42,0.12);
        glBegin(GL_POLYGON);
        glVertex2f(500,1300);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(700,1500);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(900,1650);
```

```
	glVertex2f(1300,1500);
		glColor3f(0.10,0.10,0.0);
	glVertex2f(1500,1450);
	glVertex2f(1600,1300);
//	glEnd();
	glColor3f(0.62,0.42,0.12);
	glBegin(GL_POLYGON);
	glVertex2f(1400,1300);
	glVertex2f(1800,1600);
	glColor3f(0.10,0.10,0.0);
	glVertex2f(1850,1650);
	glVertex2f(1900,1600);
	glColor3f(0.10,0.10,0.0);

	glVertex2f(2000,1300);
	glEnd();
	glColor3f(0.62,0.42,0.13);
	glBegin(GL_TRIANGLES);
	glVertex2f(2000,1300);
	glColor3f(0.10,0.10,0.0);
	glVertex2f(2200,1600);
		glColor3f(0.10,0.10,0.0);
//	glColor3f(0.60,0.40,0.0);

	glVertex2f(2200,1300);
	glEnd();

	//to draw road for Night

	glPointSize(8.0);
	glColor3f(0.10,0.10,0.0);
	glBegin(GL_POLYGON);
	glVertex2f(2200,700);
	glColor3f(0.05,0.05,0.0);
	glVertex2f(2200,400);
	glColor3f(0.10,0.10,0.0);
	glVertex2f(1700,0);
	glColor3f(0.05,0.07,0.0);
	glVertex2f(1400,0);
	glEnd();
	glColor3f(.8,.7,0.0);
	glBegin(GL_LINES);
	glVertex2f(2200,550);
	glVertex2f(1600,0);
```

```
        glVertex2f(2200,560);
        glVertex2f(1590,0);
        glEnd();


}


void draw_pixel(GLint cx,GLint cy)
{       glPolygonMode(GL_FRONT,GL_FILL);

        glBegin(GL_POINTS);

        glVertex2i(cx,cy);

        glEnd();
}


void plotpixels(GLint h,GLint k,GLint x,GLint y)
{
        draw_pixel(x+h,y+k);
   draw_pixel(-x+h,y+k);
        draw_pixel(x+h,-y+k);
        draw_pixel(-x+h,-y+k);

        draw_pixel(y+h,x+k);
        draw_pixel(-y+h,x+k);
        draw_pixel(y+h,-x+k);
        draw_pixel(-y+h,-x+k);

}

void circle_draw(GLint h,GLint k,GLint r)
{
        GLint d=1-r,x=0,y=r;

        while(y>x)
        {
                plotpixels(h,k,x,y);
                if(d<0)
                        d+=2*x+3;
                else
                {
                        d+=2*(x-y)+5;
                        --y;
                }
                ++x;
        }
        plotpixels(h,k,x,y);
}

//To draw Sea for DAY

void drawsea()
{
        glColor3f(0.0,0.6,1.0);
        glBegin(GL_POLYGON);
        glVertex2f(2200,900);
        glVertex2f(2200,900);
        glColor3f(0.0,0.7,1.0);
        glVertex2f(2200,800);
        glVertex2f(2200,750);
        glVertex2f(1900,700);
        glColor3f(0.0,0.7,1.0);
        glVertex2f(1850,600);
        glVertex2f(1800,600);
        glColor3f(.0,0.8,.9);
        glVertex2f(1700,550);
        glVertex2f(1600,550);
        glColor3f(0.0,0.6,.98);
        glVertex2f(1450,400);
        glColor3f(.0,0.8,.95);
        glVertex2f(1300,300);
        glColor3f(.0,0.7,.98);
        glVertex2f(1100,250);
        glColor3f(.0,0.8,.9);
        glVertex2f(900,200);
        glColor3f(.0,0.8,.9);
        glVertex2f(700,200);

        glColor3f(.0,0.7,.9);
        glVertex2f(500,300);
        glVertex2f(400,400);
        glColor3f(0.0,0.6,.9);
        glVertex2f(300,500);
        glVertex2f(150,600);
        glColor3f(.0,0.7,.9);
```

```
            glVertex2f(0,700);
            glVertex2f(0,1300);
            glColor3f(.0,0.8,.9);
            glVertex2f(2200,1300);
            glEnd();

}

//To draw sea for NIGHT

void drawsea2()
{

            glColor3f(1.0,1.0,1.0);
            glBegin(GL_POLYGON);
            glVertex2f(2200,900);
            glColor3f(0.,0.,.0);

            glVertex2f(2200,900);
            glColor3f(.0,0.,.0);
            glVertex2f(2200,800);
            glColor3f(1.,1.,1.0);
            glVertex2f(2200,750);
            glColor3f(.0,0.,.0);
            glVertex2f(1900,700);
            glColor3f(.0,0.,.0);
            glVertex2f(1850,600);
            glColor3f(.0,0.,.0);
            glVertex2f(1800,600);

            glVertex2f(1700,550);
            glColor3f(.0,0.,.0);
            glVertex2f(1600,550);
            glColor3f(.0,0.,.0);

            glVertex2f(1450,400);
            glColor3f(.0,0.0,.0);
            glVertex2f(1300,300);
            glColor3f(.0,0.,.0);
            glVertex2f(1100,250);
            glColor3f(.0,0.0,.0);
            glVertex2f(900,200);
            glColor3f(.0,0.,.0);
            glVertex2f(700,200);
```

```
        glColor3f(.0,0.0,.0);
        glVertex2f(500,300);
        glColor3f(.0,0.0,.0);
        glVertex2f(400,400);
        glColor3f(0.0,0.,.0);
        glVertex2f(300,500);
        glColor3f(.0,0.0,.0);
        glVertex2f(150,600);
        glColor3f(.0,0.,.0);
        glVertex2f(0,700);
        glColor3f(.0,0.0,.0);
        glVertex2f(0,1300);
        glColor3f(.0,0.,.0);
        glVertex2f(2200,1300);
        glColor3f(.0,0.,.0);
        glEnd();


}



void night()
{       // to set the background color to black

        glColor3f(0.0,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(2200,1800);
        glVertex2f(0,1800);
        glVertex2f(0,0);
        glVertex2f(2200,0);
        glEnd();

        //to draw the moon

        glPointSize(9.0);
        glColor3f(1.0,1.0,1.0);

        circle_draw(1400,1600+s,100);
        circle_draw(1400,1600+s,90);
        circle_draw(1400,1600+s,70);
        circle_draw(1400,1600+s,50);
        circle_draw(1400,1600+s,30);
```

```
            circle_draw(1400,1600+s,10);
            glPointSize(5.0);



//to draw the stars

            glBegin(GL_POINTS);
            glVertex2f(900,1700);
            glVertex2f(800,1700);
            glVertex2f(700,1100);
            glVertex2f(200,1500);
            glVertex2f(1800,1700);
            glVertex2f(1700,1200);
            glVertex2f(2000,1500);
            glEnd();
//          glColor3f(.25,.25,.25);
            mountain2();
            glColor3f(.7,.7,.7);

            drawsea2();

//to draw street light for NIGHT

            glColor3f(0.10,0.10,0.0);
            glBegin(GL_POLYGON);
            glVertex2f(1400,0);
            glColor3f(0.10,0.10,0.0);
            glVertex2f(1425,0);
            glColor3f(0.10,0.10,0.0);
            glVertex2f(1425,250);
            glVertex2f(1400,250);
            glEnd();
            glColor3f(1.0,01.0,01.0);
            circle_draw(1410,280,30);
            circle_draw(1410,280,10);

            glColor3f(0.10,0.10,0.0);
            glBegin(GL_POLYGON);
            glVertex2f(1600,200);
            glColor3f(0.10,0.10,0.0);
            glVertex2f(1625,200);
            glColor3f(0.10,0.10,0.0);
            glVertex2f(1625,350);
```

```
        glVertex2f(1600,350);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(1610,380,30);
        circle_draw(1610,380,10);

        glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(1800,380);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(1825,380);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(1825,525);
        glVertex2f(1800,525);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(1810,555,30);
        circle_draw(1810,555,10);

        glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(2000,500);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(2025,500);
        glColor3f(0.10,0.10,0.0);
        glVertex2f(2025,650);
        glVertex2f(2000,650);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(2010,680,30);
        circle_draw(2010,680,10);

        //to draw tree and boat

        drawtree2();
        drawsmallboat();
        drawboat();

}

//To draw user BOAT

void drawboat()
{
        glColor3f(0.1,0.0,0.1);
        glBegin(GL_POLYGON);

        glVertex2f(700+n,950);
        glColor3f(.6,0.5,0.0);
        glVertex2f(800+n,800);
        glVertex2f(1200+n,800);
        glColor3f(0.1,0.0,0.1);
        glVertex2f(1400+n,950);
        glEnd();

    glColor3f(.6,0.5,0.0);
        glBegin(GL_TRIANGLES);
        glVertex2f(1200+n,950);

        glColor3f(.6,0.5,0.5);
        glVertex2f(1350+n,950);
        glColor3f(.0,0.2,0.6);
        glVertex2f(1000+n,1150);
        glEnd();
        glPointSize(9.0);
        glBegin(GL_LINES);
        glVertex2f(1000+n,950);
        glVertex2f(1000+n,1150);
        glEnd();

        glColor3f(0.1,0.0,0.1);
        glBegin(GL_POLYGON);
        glVertex2f(850+n,950);
        glVertex2f(1000+n,950);
        glVertex2f(1000+n,1000);
        glVertex2f(850+n,1000);
        glEnd();
}
void redrawboat1()
{       if(boat==0)
        {
                if(n<1600)
                {

                        n+=2;
                        glutPostRedisplay();
                }
                else
                {       n=n-2400;
                glutPostRedisplay();
                }
        }
}


void drawsmallboat()
{
        glColor3f(0.0,1.0,0.0);

        glBegin(GL_POLYGON);

        glVertex2f(400+bi,1200);
                glColor3f(1.0,0.6,.9);
        glVertex2f(475+bi,1100);
        glColor3f(0.3,0.0,0.2);
        glVertex2f(575+bi,1100);
        glVertex2f(650+bi,1200);
        glEnd();

        glBegin(GL_TRIANGLES);
        glVertex2f(550+bi,1200);
        glColor3f(1.0,0.6,.9);

        glVertex2f(650+bi,1200);
        glColor3f(.0,0.2,0.6);
        glVertex2f(500+bi,1250);
        glEnd();
        glPointSize(9.0);
        glBegin(GL_LINES);
        glVertex2f(500+bi,1250);
        glVertex2f(500+bi,1200);
        glEnd();

}

//To loop small boat

void redrawboat()
{       if(boat==0)
        {
                if(bi<1600)
                {

                        bi+=2;
                        glutPostRedisplay();
                }
                else
                {       bi=bi-2400;
                glutPostRedisplay();
                }
        }
}

// Keyboard function

void key(unsigned char key,int x,int y)
{
        if(key=='x')
        {

          n=n+10;
          if(n>2000) n=-800;
    glutPostRedisplay();
        }
        if(key=='y')
        {
                n=n-10;
                if(n<-1400) n=1400;
        glutPostRedisplay();
        }
        if(key=='z')
        {        s=s+10;
                glutPostRedisplay();
        }
        if(key=='w')
        {       s=s-10;
                glutPostRedisplay();
        }
        if(key=='s') //to stop boat
        {       boat=1;
                glutPostRedisplay();
        }
        if(key=='r') // to restart boat
        {       boat=0;
                glutPostRedisplay();
        }

}


//To draw tree for DAY

void drawtree()
{
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_POLYGON);
        glVertex2f(0,300);
        glVertex2f(100,400);
        glVertex2f(40,400);
                glColor3f(0.30,0.4,0.);
        glVertex2f(150,500);
        glVertex2f(90,500);
        glVertex2f(200,600);
        glColor3f(0.10,0.4,0.);
        glVertex2f(300,500);
        glVertex2f(250,500);

        glVertex2f(350,400);
        glVertex2f(300,400);
        glVertex2f(350,300);
        glEnd();

        glColor3f(0.0,0.2,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(400,250);
        glVertex2f(450,350);
//      glVertex2f(400,350);
//      glVertex2f(450,450);
        glColor3f(0.1,0.4,0.);
        glVertex2f(400,450);
        glVertex2f(500,650);
        glColor3f(0.30,0.4,0.);
        glVertex2f(650,450);
        glVertex2f(600,450);
        glColor3f(0.0,0.3,0.2);
        glVertex2f(700,350);
        glVertex2f(650,350);
        glColor3f(0.30,0.4,0.);
        glVertex2f(700,250);
        glEnd();

        glColor3f(0.0,0.3,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(750,200);
        glVertex2f(850,300);
        glColor3f(0.30,0.4,0.);

        glVertex2f(800,300);

        glVertex2f(850,400);
        glVertex2f(800,400);
        glColor3f(0.0,0.2,0.1);
        glVertex2f(850,500);
        glColor3f(0.0,0.2,0.0);
        glVertex2f(800,500);
        glVertex2f(900,600);
        glColor3f(0.1,0.4,0.);
        glVertex2f(1000,500);
        glVertex2f(950,500);
        glVertex2f(1100,400);
        glColor3f(0.2,0.4,0.0);
        glVertex2f(1050,400);
        glVertex2f(1150,300);
        glColor3f(0.2,0.5,0.0);
        glVertex2f(1100,300);
        glVertex2f(1150,200);
        glEnd();


//stamp
        glColor3f(.2,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(150,0);
//      glColor3f(0.47,0.0,0.0);
        glVertex2f(150,300);
        glColor3f(0.57,0.0,0.0);
        glVertex2f(200,300);
        glColor3f(0.47,0.0,0.0);
        glVertex2f(200,0);
        glEnd();


        glColor3f(0.2,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(500,0);
        glVertex2f(500,250);
        glColor3f(0.57,0.0,0.0);
        glVertex2f(550,250);
        glVertex2f(550,0);
        glEnd();

        glColor3f(0.2,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(950,0);
        glVertex2f(950,200);
        glColor3f(0.57,0.0,0.0);
        glVertex2f(1000,200);
        glVertex2f(1000,0);
        glEnd();
}

//To draw tree for NIGHT

void drawtree2()
{
        glColor3f(0.1,0.1,.0);
        glBegin(GL_POLYGON);
        glVertex2f(0,300);

        glVertex2f(100,400);
        glColor3f(0.1,0.1,.0);
        glVertex2f(40,400);
        glColor3f(0.,0.,.0);
        glVertex2f(150,500);
        glVertex2f(90,500);
        glVertex2f(200,600);
        glColor3f(0.1,0.1,.0);
        glVertex2f(300,500);
        glVertex2f(250,500);
        glColor3f(0.,0.,.0);
        glVertex2f(350,400);
        glVertex2f(300,400);
        glVertex2f(350,300);
        glEnd();


        glColor3f(0.1,0.1,.0);
```

```
glBegin(GL_POLYGON);
glVertex2f(400,250);
glVertex2f(450,350);
glColor3f(0.,0.,.0);
glVertex2f(400,450);
glVertex2f(500,650);
glColor3f(0.1,0.1,.0);
glVertex2f(650,450);
glVertex2f(600,450);
glColor3f(0.,0.,.0);
glVertex2f(700,350);
glVertex2f(650,350);
glColor3f(0.1,0.1,.0);
glVertex2f(700,250);
glEnd();


glBegin(GL_POLYGON);
glVertex2f(750,200);
glVertex2f(850,300);
glColor3f(0.,0.,.0);
glVertex2f(800,300);
glColor3f(0.1,0.1,.0);
glVertex2f(850,400);
glVertex2f(800,400);

glVertex2f(850,500);
glColor3f(0.1,0.1,.0);

glVertex2f(800,500);
glVertex2f(900,600);
glColor3f(0.,0.,.0);
glVertex2f(1000,500);
glVertex2f(950,500);
glVertex2f(1100,400);
glColor3f(0.1,0.1,.0);
glVertex2f(1050,400);
glVertex2f(1150,300);
glColor3f(0.,0.,.0);
glVertex2f(1100,300);
glVertex2f(1150,200);
glEnd();
```

```
//stamp
        glColor3f(.2,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(150,0);
        glColor3f(0.,0.,.0);
        glVertex2f(150,300);
        glColor3f(0.1,0.0,0.0);
        glVertex2f(200,300);
        glColor3f(0.2,0.0,0.0);
        glVertex2f(200,0);
        glEnd();


        glColor3f(0.2,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(500,0);
        glVertex2f(500,250);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(550,250);
        glColor3f(0.1,0.0,0.0);
        glVertex2f(550,0);
        glEnd();

        glColor3f(0.2,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(950,0);
        glVertex2f(950,200);
        glColor3f(0.0,0.0,0.0);
        glVertex2f(1000,200);
        glColor3f(0.1,0.0,0.0);
        glVertex2f(1000,0);
        glEnd();
}


void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0,1.0,1.0);
        if(glob==1)
                night();
        if(glob==0)
                day();
```

```
   redrawboat();
   redrawboat1();

            glFlush();
            glutSwapBuffers();

}
void day()
{           //to set the sky color and ground color

            glColor3f(.8,.37,0.0);

            glBegin(GL_POLYGON);
            glVertex2f(2200,1800);
            glColor3f(.7,.37,0.0);
            glVertex2f(0,1800);
            glColor3f(.4,.6,0.0);
            glVertex2f(0,0);
                        glColor3f(.4,.6,0.0);
            glVertex2f(2200,0);
            glEnd();

            // to draw the mountains
            mountain();

// to draw the sea
            drawsea();

            // to draw the sun

            glPointSize(9.0);
            glColor3f(.9,0.2,0.0);
            circle_draw(1400,1600,100);
            glColor3f(.95,0.2,0.0);
            circle_draw(1400,1600,90);
            glColor3f(1.0,0.25,0.0);
            circle_draw(1400,1600,70);
                  glColor3f(1.1,0.3,0.0);
            circle_draw(1400,1600,50);
                  glColor3f(1.15,0.35,0.0);
            circle_draw(1400,1600,30);
                  glColor3f(1.18,.38,0.0);

            circle_draw(1400,1600,10);
```

```
        drawtree();

//to draw street light for DAY

                glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(1400,0);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1425,0);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1425,250);
        glVertex2f(1400,250);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(1410,280,30);
        circle_draw(1410,280,10);

                        glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(1600,200);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1625,200);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1625,350);
        glVertex2f(1600,350);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(1610,380,30);
        circle_draw(1610,380,10);

                        glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(1800,380);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1825,380);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1825,525);
        glVertex2f(1800,525);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(1810,555,30);
        circle_draw(1810,555,10);
```

```
                        glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(2000,500);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(2025,500);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(2025,650);
        glVertex2f(2000,650);
        glEnd();
        glColor3f(1.0,01.0,01.0);
        circle_draw(2010,680,30);
        circle_draw(2010,680,10);
        drawsmallboat();
        drawboat();
                //to draw road DAY

        glPointSize(8.0);
                glColor3f(0.10,0.10,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(2200,700);
                glColor3f(0.05,0.05,0.0);
        glVertex2f(2200,400);
                glColor3f(0.10,0.10,0.0);
        glVertex2f(1700,0);
        glColor3f(0.05,0.07,0.0);
        glVertex2f(1400,0);
        glEnd();
                glColor3f(1.0,1.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(2200,550);
        glVertex2f(1600,0);
        glVertex2f(2200,560);
        glVertex2f(1590,0);
        glEnd();

}


void mainmenu(int id)
{
        switch(id)
        {
                case 1:exit(0);
                break;
                case 2:glob=0;
                        break;
                case 3:glob=1;
                        break;

        }
        glutPostRedisplay();
}

int main(int argc,char** argv)
{

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(2200,1800);
        glutInitWindowPosition(0,0);
        glutCreateWindow("day/night seaview");

        glutKeyboardFunc(key);
        glutDisplayFunc(display);

        glutCreateMenu(mainmenu);
        glutAddMenuEntry("QUIT",1);
        glutAddMenuEntry("DAY VIEW",2);
        glutAddMenuEntry("NIGHT VIEW",3);

        glutAttachMenu(GLUT_RIGHT_BUTTON);
        init();
        glutMainLoop();
        return 0;
}
```