**Chapter 1**

**Preamble**

* 1. **Introduction**

**COMPUTER GRAPHICS:**

Computer Graphics is concerned with all aspect of producing pictures or image using computer. The field began humble almost 50 years ago, with the display of few lines on the cathode-ray tube(CRT); now, we can create image using computer that are indistinguishable from photographs from the real objects. We routinely train pilots with simulated airplane, generating graphical display of the virtual environment in the real time. Feature length movies made entirely by computer have been successful, both critically and financially; massive multiplayer game can involve tens of thousands of concurrent participants.

Graphics is created using computers and, more generally, the representation and manipulation of pictorial data by a computer. The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. The phrase “Computer Graphics” was coined in 1960 by William Fetter, a graphic designer for Boeing.

In today’s world advanced technology, interactive computer graphics has become a powerful tool for the production of realistic features. Today’s we find computer graphics used in various areas that include science, engineering, medicine, business, industry, art, entertainment etc. The main reason for effectiveness of the interactive computer graphics is the speed with which the user can understand the displayed information.

The graphics in openGL provides a wide variety of built-in function. The computer graphics remains one of the most exciting and rapidly growing computer fields. It has become a common element in user interface, data visualization, TV commercials, motion picture and many other applications. The current trend of computer graphics is to incorporate more physics principles into 3D graphics algorithm to better simulate the complex interactions between objects and lighting environment.

**1.1.1 History of Computer Graphics**

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. This was started with the display of data on plotters and CRT. Computer Graphics is also defined as the study of techniques to improve the communication between user and machine, thus Computer Graphics is one of the most effective medium of communication between machine and user.

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeng. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

* + 1. **Introduction to Open GL**

OpenGL has become a widely accepted standard for developing graphics application. OpenGL is easy to learn, and it possesses most of the characteristics of other popular graphics system. It is top-down approach. OpenGL is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.

OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms.

The interface between the application program and the graphics system can be specified through that set of function that resides in graphics library. The specification is called the APPLICATION PROGRAM INTERFACE (API). The application program sees only the API and is thus shielded from the details both the hardware and software implementation of graphics library. The software driver is responsible for interpreting the output of an API and converting these data to a form that is understood by the particular hardware.

Most of our applications will be designed to access openGL directly through functions in three libraries. Function in the main GL library have name that begin with the letter gl and stored in the library. The second is the openGL utility Library (GLU). This library uses only GL function but contains codes for creating common object and viewing. Rather then using an different library for each system we used available library called openGL utility toolkit (GLUT). It is used as #include<glut.h>

A graphics editor is a computer program that allows users to compose and edit pictures interactively on the computer screen and save them in one of many popular “bitmap” or “raster” a format such as TIFF, JPEG, PNG and GIF.

Graphics Editors can normally be classified as:

2D Graphics Editors.

3D Graphics Editors.

A 3D Graphics Editor is used to draw 3D primitives Rectangles, Circle, polygons, etc and alter those with operations like cut, copy, paste. These may also contain features like layers and object precision etc.

3D Graphics Editor should include the following features:

Facilities: Cursor Movement, Editing picture objects.

Good User Interface: GUI / Toolbars / Icon based User Interface.

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. A particular graphics software system called OpenGL, which has become a widely accepted standard for developing graphics applications .

The applications of computer graphics in some of the major areas are as follows

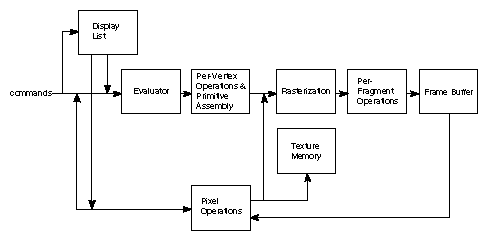
1. Display of information.

2. Design.

3. Simulation and Animation.

4. User interfaces.

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.



**Figure 1.1**: OpenGL Block Diagram

We can choose to accumulate some of commands in a *display list* for processing at a later time. The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. *Rasterization* produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed into the last stage, *per-fragment operations*, which perform the final operations on the data before it's stored as pixels in the *frame buffer.*

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

My project named “GRAPHICAL IMPLEMENTATION OF LANDSCAPE” uses OpenGL software interface and develops 2D images. This project uses the techniques like Translation, motion, display list, transformation techniques, etc.

* 1. **Objectives**

Objectives of our project are

* To design model using C graphical functions to demonstrate the scenery of nature
* Developing a package using computer graphics with OpenGL.
* Migration from text editor to OpenGL.
* To make the boat move forward and backward
* To on and off the fountain
* To move the moon in upward and downward directions
* To change the color of the fountain
  1. **Organization of the Report**

Chapter 1 provides the information about the basics of OpenGL. In Chapter 2, all the OpenGL functions used in our program is described. Chapter 3 gives the idea of the project and its actual implementation. Chapter 4 discusses about the testing and limitations of the program. Chapter 5 concludes by giving the direction for future enhancement.

**1.4 Summary**

The chapter discussed before is an overview about the computer graphics, its history and OpenGL interface. It even includes the OpenGL block diagram. The scope of study and objectives of the project are mentioned clearly. The organization of the report is been pictured to increase the readability. Further, coming up chapter depicts the OpenGL built-in functions used in project source code.

**Chapter 2**

**SPECIFICATIONS & REQUIREMENTS**

**Hardware requirements**:

Pentium or higher processor.

16 MB or more RAM.

A standard keyboard, and Microsoft compatible mouse

VGA monitor.

If the user wants to save the Created files a secondary storage medium can be Used.

**Software requirements**:

The graphics package has been designed for OpenGL; hence the machine must have geditor.

Turbo c Libraries are used and hence a TC version 2 or later is required.

**Development platform**: Ubuntu 16.04.

**Chapter 3**

**A Preview of OpenGL Functions**

When we start drawing any graphics in OpenGL using C language we need a header file called <GL/glut.h>. The header file contains definitions and explanations of all functions and constants we’ll need, whereas the graphics functions are kept in the library file. Both these files are provided as a part of TURBO C compliers.

* **stdio.h:**

This is a standard input header file which is used in any program. This file contains all the built-in functions like printf(),scanf(),fopen(),fclose() etc.It also contains data types and global variables. Some of the examples are BUFSIZ, EOF, and NULL etc.

* **stdlib.h:**

This is also one the standard library header file which contains the entire standard library functions like exit, free, alloc, mallocetc.Some of the constants and data types are NULL,size\_t etc.

* **Gl/glut.h :**

This is very familiar library function of visual basic graphics library. This header file contains so many numbers of built in functions of a graphics library.

The different OpenGL functions used in our project are described as follows:

* **Name:**glBegin()

**Cspecification**: glBegin(glEnum mode)

**Description**: Initiates a new primitive of type mode and starts collection of vertices. Values of mode include GL\_POINTS, GL\_LINES and GL\_POLYGON

* **Name:**glEnd()

C **specification**: void glEnd()

**Description**: Terminates a list of vertices.

* **Name:**glPushMatrix()

**Cspecification**: void glPushMatrix( )

**Description**: Pushes and pops from the attributes stack.

* **Name:**glPopMatrix( )

**Cspecification**: void glPopMatrix( )

**Description**: All attributes are poped from the stack.

* **Name:** glutInit( )

**Cspecification**: void glutInit ( )

**Description**: All Initializes the GLUT. The arguments from main are passed in and can be used by the application.

* **Name:**glutCreateWindow( )

**Cspecification**: void glutCreateWindow( )

**Description**: creates a window on the display. The string title can be used to label the window.

* **Name:**glutDisplayMode()

**CSpecification**: void glutDisplayMode()

**Description**: Requests a display with the properties in mode. The mode is determined by the logical OR of options including the color model.

* **Name:**glutWindowSize( )

**CSpecification**: void glutWindowSize( )

**Description**: Specifies the initial height and width of the window in pixels.

* **Name:** glutInitwindowPosition ()

**CSpecification**: void glutInitwindowPosition(int x, int y)

**Description**: Specifies the initial position of the top-left corner of the window in pixels.

* **Name:**glutMainLoop()

**CSpecification**: Void glutMainLoop()

**Description**: Cause the program to enter an event-processing loop.It should be the last statement main.

* **Name**: glutDisplayFunc()

**C Specification**: Void glutDisplayFunc(void(\*func)(void))

**Description**: registers the display functions func that is executed when the window must to be redrawn.

* **Name:**glutPostRedisplay()

**CSpecification:** void glutPostRedisplay()

**Description**: Requests that the display callback be executed after the current callback

returns.

* **Name:**glMatrixMode( )

**C Specification**: void glMatrixMode(GLenum mode)

**Description**: Specify matrix will be affected by subsequent transformation mode can be GL\_MODELVIEW, GL\_PROJECTION.

* **Name:**glTranslatef( )

C **specification:** void glTranslatef(GLfloat*x*, GLfloat*y*, GLfloat*z* )

**Description**: glTranslatef produces a translation by (x, y, z).

* **Name**: glut keyboard Func( )

**Cspecification**: void key (unsigned char keys, int x, int y)

**Description**: sets the keyboard interaction with the current window.

* **Name:**glClear()

**Cspecification:**glClear(GL\_COLOR\_ BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT**);**

**Description:** to refresh the color buffer and depth buffer so that ifalgorithm stores information in the depth buffer, we must clear this buffer whenever we wish to redraw the display.

* **Name:**glutSwapBuffers()

**C specification:** glutSwapBuffers()

**Description:** Swaps the buffers of the current window if double buffered Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

* **Name:** glFlush()

**C specification:** glFlush()

**Description:**the call to glFlush ensures that points are rendered to the screen.

* **Name:** glutIdleFunc ()

**C specification**: voidglutIdleFunc (void (\*f) (void))

**Description:**It is used to register the display callback function ‘f’ that is executed whenever there are no other events to be handled

* **Name:** glOrtho()

**Cspecification:**voidglOrtho(GLdoubleleft,GLdoubleright, GLdoublebottom,GLdoubletop,GLdoublenear,GLdouble far)

**Description:** This function defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

* **Name:** glutCreateSubWindow()

**Cspecification**: int glutCreateSubWindow(int win, int x, int y, int width, int height);

**Description:** This function creates a subwindow of the window identified by win of

size width and height at location x and y within the current window. Implicitly,

the current window is set to the newly created subwindow.

**Chapter 4**

**System Design and Implementation**

**4.1 Introduction**

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development.

This Project is implemented using OpenGl, which is proven to be a very efficient tool in the field of computer graphics, programming is done under Ubuntu 6.0.4 platform. Glut.h library is used to create the objects and to translate them. C++ programming language is used to implement the entire code. Interface to the program is provided with the help of input device keyboard and mouse.

The objects in the project such as Sea, Mountain, Trees, Boat, Road, Street lights, Sun and Moon are created by using **primitive** function GL\_**POLYGON**. The fountain is drawn making use of **displaylists().**The FLOWING FOUNTAIN depicts a model of a fountain through which water is continuously flowing out in its idle state. The user is provided with an option to change the color of the fountain using the RIGHT MOUSE BUTTON. The water flows out through different levels in the fountain, giving it a realistic look. User can specify these levels as three, four or five at the beginning.

Human interface to the same is provided by keyboard function and Mouse pop up menus.

**4.2 Overall Design Process**

**Movement of a drop in fountain**:

The movement of a drop contains two factors.

The direction, how the drop gets out of the fountain and the gravity. The position of a drop is pretty easy to compute if we know, how much time has passed since the drop has leaved the fountain.

We have to multiplicate the vector of the constant moving (how the drop leaves the fountain) with the time and then subtract the squared time multiplicated with an acceleration factor. This acceleration factor contains the weight of a drop and the power of gravity. We now have to know the direction, how the drop comes out of the fountain, but this is just a bit calculating with sine and cosine.

Blending means that a pixel on the screen isn't replaced by another one, but they are "mixed". Therefore you can use the alpha value of colors, it indicates how much of the color of the consisting pixel is used for the new color - for antialiasing of points, OpenGL computes this alpha value.

After calling *glEnable(GL\_BLEND);* you have to tell OpenGL how to use the alpha values. It isn't specified, that a higher alpha-value means more transparency or something like that. You can use them as you want. To tell OpenGL \_what\_ you want, you must use *glBlendFunc().* It takes two parameters, one for the source factor and the second for the destination factor. I used GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA as parameters. This is quite an often used combination and affects, that the higher the alpha value, the less transparency of the incoming fragment

**4.2.1 User defined functions:**

* **Name:** key()

**Cspecification**: void key()

**Description**: To declare the function that indicates the usage of different keys.

* **Name:** reshape()

**Cspecification**: void reshape()

**Description**: to reshape.

* **Name:** redrawboat()

**Cspecification**: void redrawboat()

**Description**: to the continuous movement of the boat.

* **Name:** day()

**Cspecification**: void day()

**Description**: used to render the scenery in day mode.

* **Name:** night()

**Cspecification**: void night()

**Description**: used to render the scenery in night mode.

* **Name:** drawfountain()

**Cspecification**: void drawfountain()

**Description**: To draw the fountain.

* **Name:** randcolor()

**Cspecification**: void randcolor()

**Description**: used to get different colors for fountain.

* **Name:** drawtextxy()

**Cspecification**: void drawtextxy()

**Description**: used to print the characters on the screen.

* **Name:** display()

**Cspecification:** void display()

**Description**: To display the window.

**4.2.2 ALGORITHMS:**

1. //To loop small boat

//input: key ‘r’ from the keyboard

//output: movement of the boat

if(boat==0) then

{

if(bi<1600) then

{

bi+=2;

glutPostRedisplay();

}

else

{ bi=bi-2400;

glutPostRedisplay();

}

}

1. //For the continuous motion of the fountain

//output: continuous flow of fountain

FountainDrops = new CDrop [ (int)DropsComplete ];

FountainVertices = new SVertex [ (int)DropsComplete ];

SVertex NewSpeed;

GLfloat DropAccFactor;

GLfloat TimeNeeded;

GLfloat StepAngle;

GLfloat RayAngle;

GLint i,j,k;

for (k = 0; k <Steps; k++)

{

for (j = 0; j < RaysPerStep; j++)

{

for (i = 0; i < DropsPerRay; i++)

{

DropAccFactor = AccFactor + GetRandomFloat(0.0005);

StepAngle = AngleOfDeepestStep + (90.0-AngleOfDeepestStep)

\* GLfloat(k) / (Steps-1) + GetRandomFloat(0.2+0.8\*(Steps-k-1)/(Steps-1));

NewSpeed.x = cos ( StepAngle \* PI / 180.0) \* (0.2+0.04\*k);

NewSpeed.y = sin ( StepAngle \* PI / 180.0) \* (0.2+0.04\*k);

RayAngle = (GLfloat)j / (GLfloat)RaysPerStep \* 360.0;

NewSpeed.z = NewSpeed.x \* sin ( RayAngle \* PI /180.0);

NewSpeed.x = NewSpeed.x \* cos ( RayAngle \* PI /180.0);

TimeNeeded = NewSpeed.y/ DropAccFactor;

FountainDrops[i+j\*DropsPerRay+k\*DropsPerRay\*RaysPerStep].SetConstantSpeed ( NewSpeed );

FountainDrops[i+j\*DropsPerRay+k\*DropsPerRay\*RaysPerStep].SetAccFactor (DropAccFactor);

FountainDrops[i+j\*DropsPerRay+k\*DropsPerRay\*RaysPerStep].SetTime(TimeNeeded \* i / DropsPerRay);

}

}

}

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer( 3,GL\_FLOAT,0,FountainVertices);

3)//for the movement of boat

//input: keys ‘x’ and ‘y’ from the keyboard

//output: boat moving in forward or backward direction

if(key=='x') then

{

bi=bi+10;

if(bi>2000) bi=-800;

glutPostRedisplay();

}

if(key=='y') then

{

bi=bi-10;

if(bi<-1400) bi=1400;

glutPostRedisplay();

}

4)// for the movement of moon

//input: keys ‘z’ and ‘w’ from the keyboard

//output: moon moving in upward or downward direction

if(key=='z') then

{ s=s+10;

glutPostRedisplay();

}

if(key=='w') then

{ s=s-10;

glutPostRedisplay();

}

5)//To On the fountain

//input: key ‘i’ from the keyboard

//output: we can see the flowing fountain on the screen

if(key=='i') then

{

ww=glutCreateSubWindow(id,200,350,200,200);

glEnable(GL\_DEPTH\_TEST);

InitFountain();

glutDisplayFunc(Display);

glutReshapeFunc(Reshape);

glutCreateMenu(colour);

glutAddMenuEntry("RANDOM",1);

glutAddMenuEntry("GREEN",2);

glutAddMenuEntry("PINK",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

6) //To Off the fountain

//input: key ‘j’ from the keyboard

//output: disappearance of flowing fountain on the screen

if(key=='j') then

{

glutDestroyWindow(ww);

}

**4.3 Flowchart**

false true yes no yes no yes no yes

**On and Off of fountain**

**Movement of the moon**

**Movement of the boat**

**Update display window**

**else If(key==I ||key==j)**

**else If(key==z ||key==w)**

**If(key==x ||key==y )**

**While**

**(option !=”quit”)**

**Display Window**

**Figure 4.1**: Flowchart

**Chapter 5**

**Results and Discussions**

The project is compiled and executed on g++. We have put in few screen shots in here to show the working of our project.

**Snapshots:**

Figure 5.1: Initially cover page will be displayed.



Figure 5.2: After pressing enter key the scenery is displayed as below.

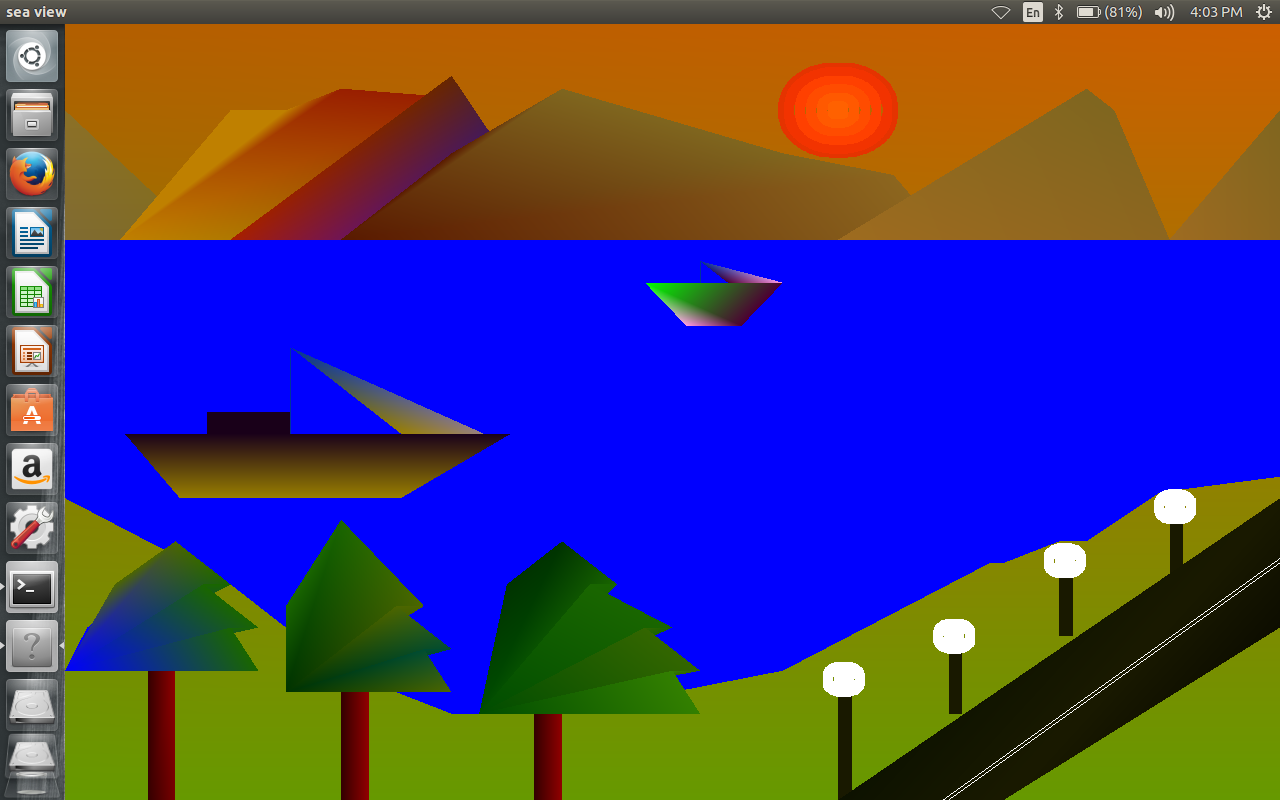


Figure 5.3: Day mode with fountain On

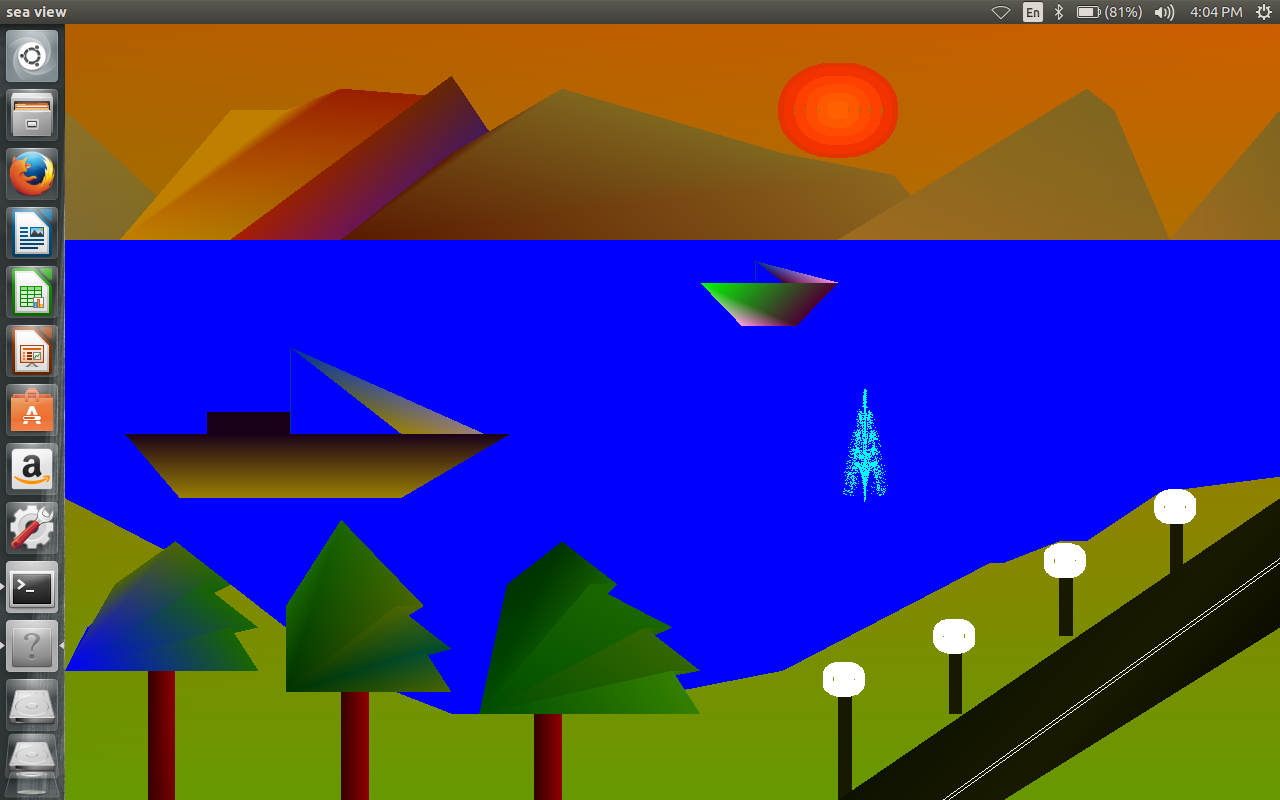


Figure 5.4: Day mode with moving boat.

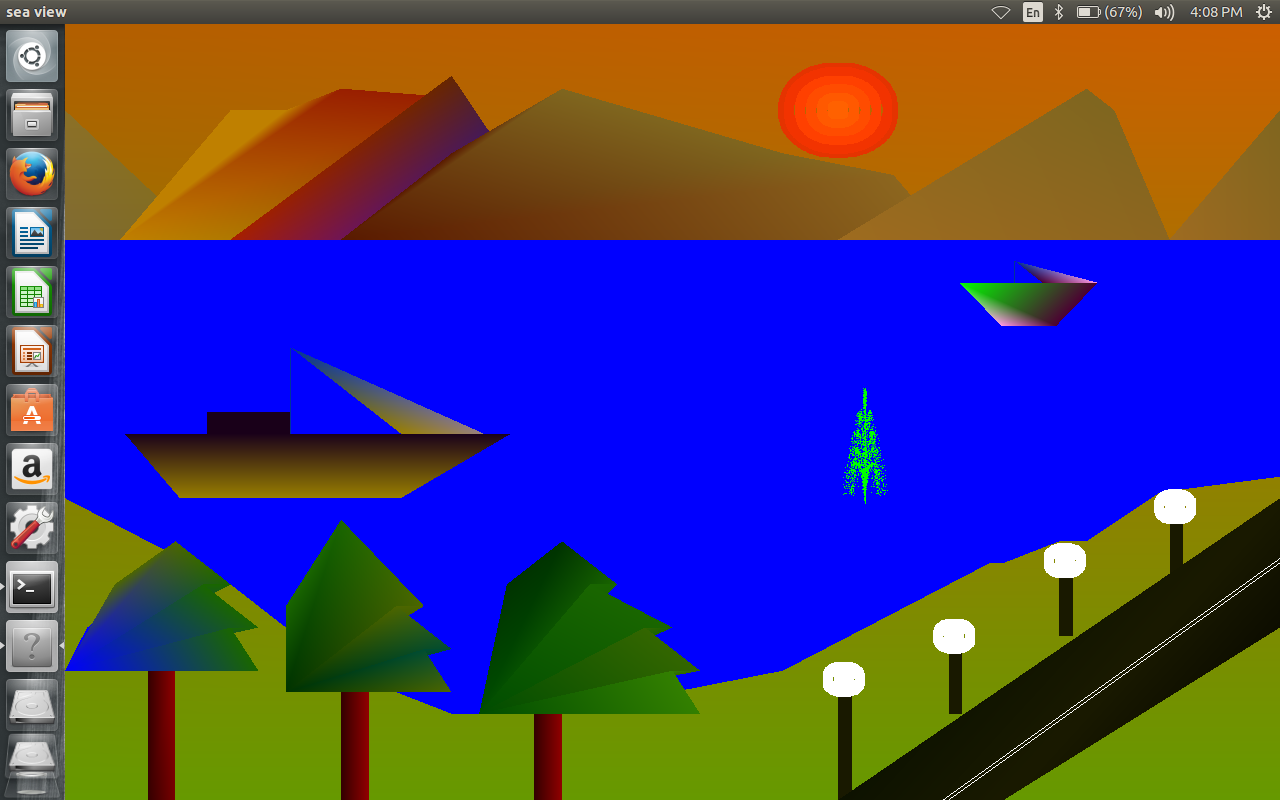


Figure 5.5: Display with pop-up menus.

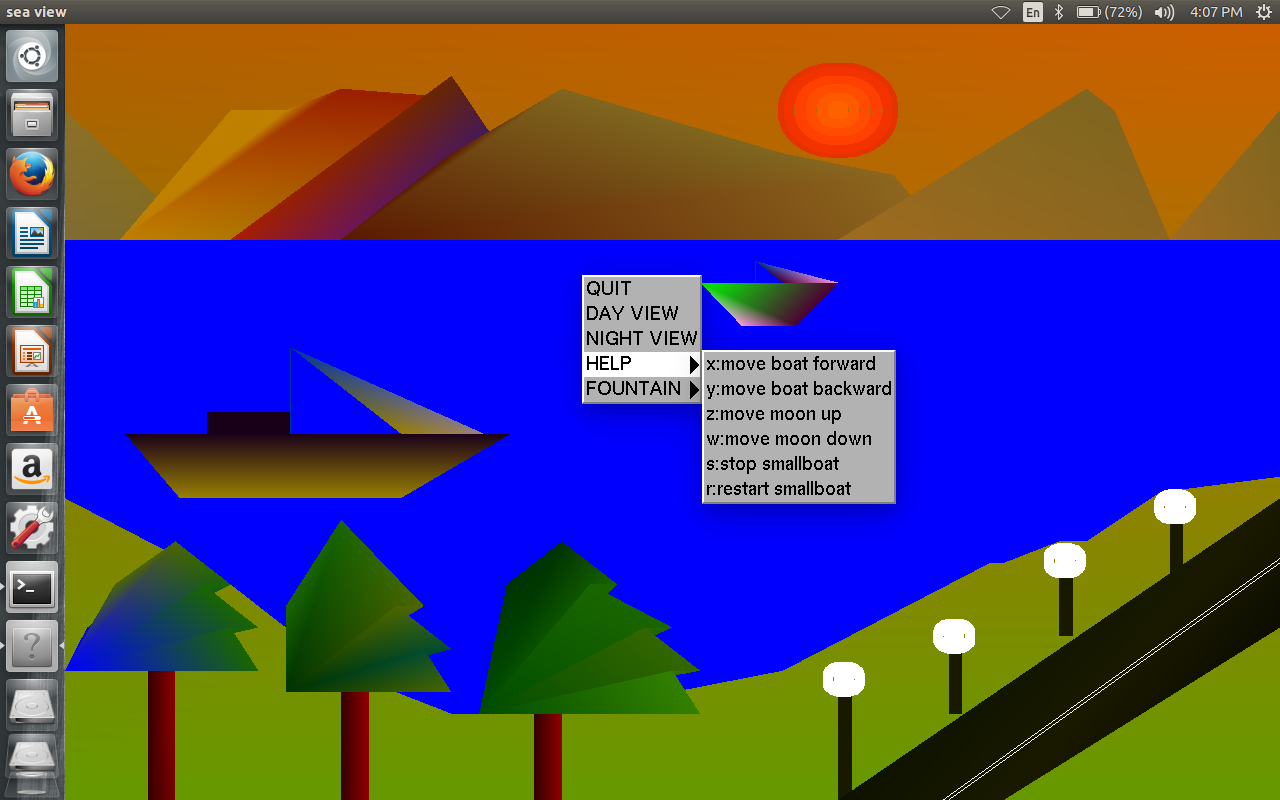


Figure 5.6: Menu for fountain to change the color

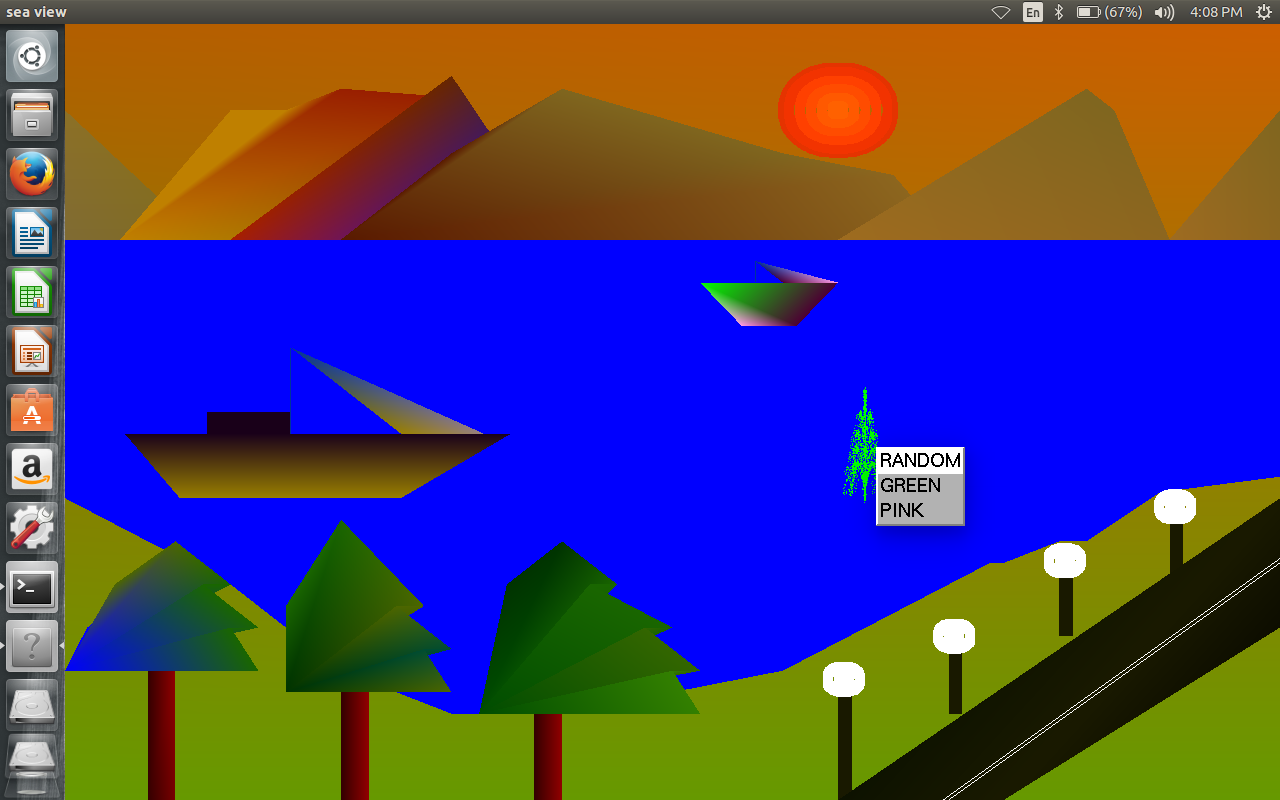
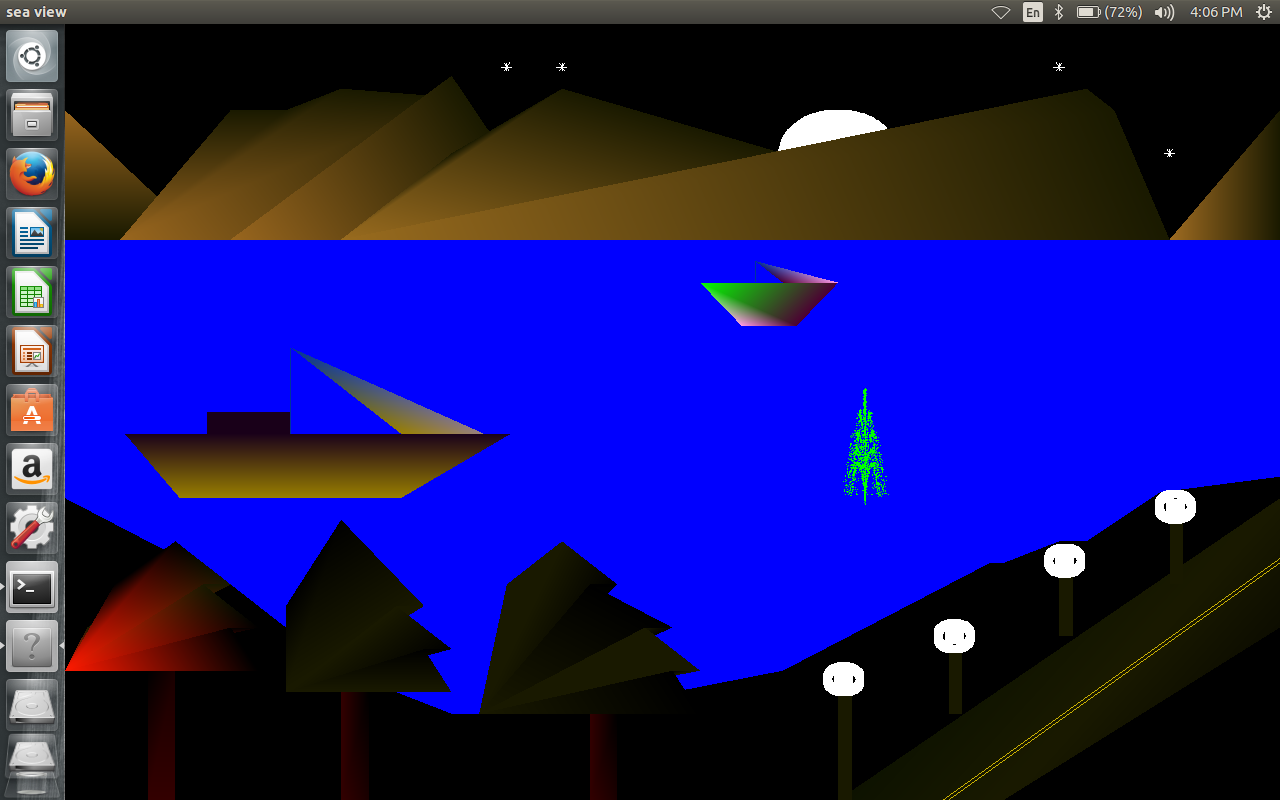
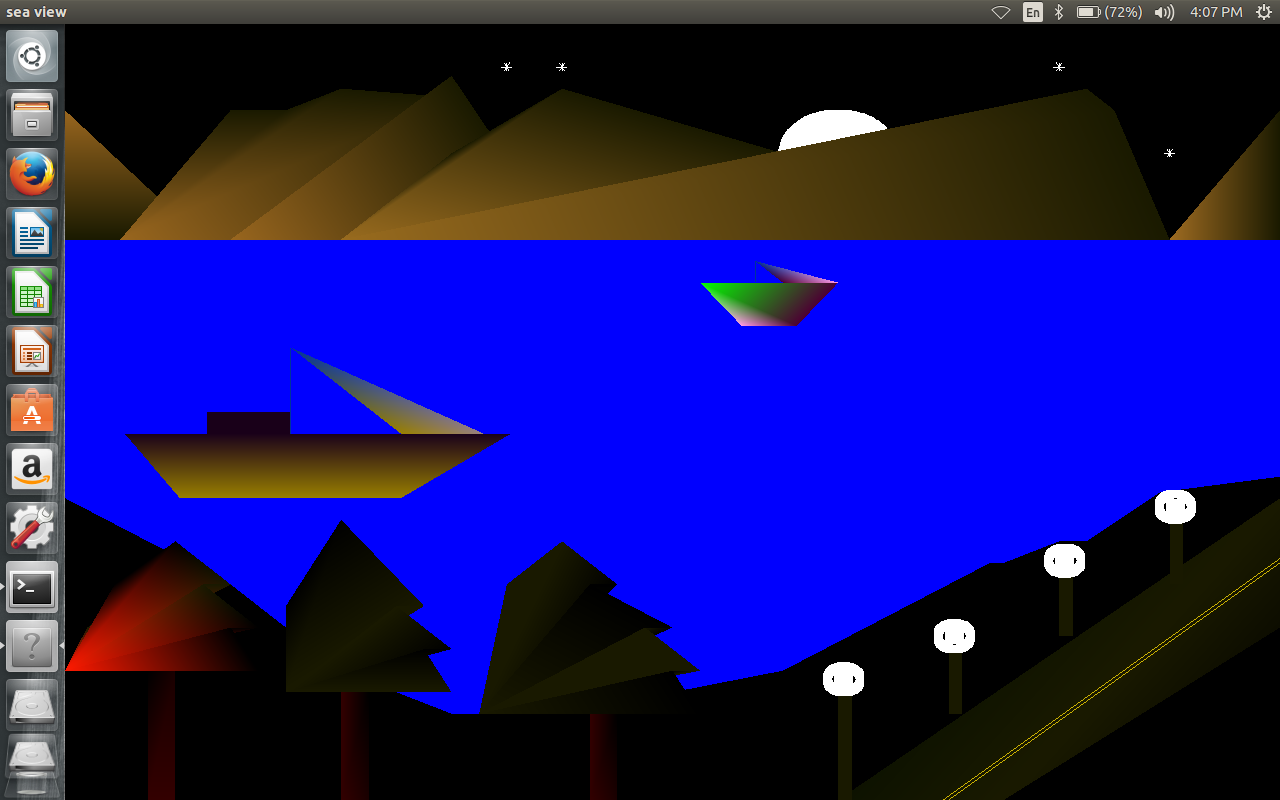


Figure 5.7: Night mode with fountain

Figure 5.8: Change of color in the fountain



**Figure 5.9:** Night mode with the movement of moon



**Chapter 6**

**Conclusion and Future Scope**

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily.

The development of the mini project has given us a good exposure to OpenGL by which we have learnt some of the technique which help in development of animated pictures, gaming.

Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

**References**

[1] Edward Angel, “Interactive Computer Graphics”, Pearson education,

Fifth Edition.

[2] <https://www.opengl.org> > spec3 > node17

[3] [www.OpenGL.org/recources/code/samples](http://www.OpenGL.org/recources/code/samples)

**Appendix A**

**OpenGL Routines:**

* **void glBegin (glEnum mode) :**

Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL\_POINTS, GL\_LINES, and POLYGON

* **void glEnd () :**

Terminate a list of vertices.

* **void glutInitWindowPositin(intx,int y) :**

Specify the initial position of the top-left corner of the window in pixel.

* **void glutMainLoop() :**

Cause the program to enter an event-processing loop. It should be the last statement in Main

* **void glutDisplayFunc(void (\*func)(void)) :**

Registers the display functions func that is executed when the window needs to be redrawn.

* **void glutSwapBuffers() :**

Swaps the front and back buffers.

* **void glutAddmenuEntry (char \*name, int value) :**

Add the entry with the string name displayed to the current menu. Values are returned to the menu callback when the entry is selected.

* **void glMatrixMode (GLenum mode) :**

Specifies matrix will be affected by subsequent transformations. Mode can be GL\_MODELVIEW,GL\_PROJECTION

* **void LoadIdentity() :-**

Set the current transformation matrix to an identity matrix.