**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction to Computer Graphics**

Computer Graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computer themselves. Now it has grown to that extent that it has included the creation, storage and manipulation of models and images of objects. Computer Graphics today is largely interactive; it is the user who controls the contents, structure and appearance of objects and of their displayed images by using input devices, such as keyboard, mouse or touch-sensitive panel on the screen. Although early applications in engineering and science had to rely on expensive and cumbersome equipment, advance in computer technology have made interactive graphics as a practical tool. Computer Graphics is an integral part of all computer user interfaces, and is indispensable for visualizing two dimensional, three-dimensional, and higher-dimensional objects: areas as diverse such as education, science, Computer Graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computer themselves. Now it has grown to that extent that it has included the creation, storage and manipulation of models and images of objects. Computer Graphics today is largely interactive; it is the user who controls the contents , structure and appearance of objects and of their displayed images by using input devices, such as keyboard, mouse or touch-sensitive panel on the screen. Although early applications in engineering and science had to rely on expensive and cumbersome equipment, advance in computer technology have made interactive graphics as a practical tool. Computer Graphics is an integral part of engineering, medicine, commerce, the military; advertising and entertainment all rely on computer graphics.

Until the early 1980’s, Computer Graphics was a small, specialized field, largely because the hardware was expensive and graphics-based application programs that were easy to use and cost effective were few. Then, personal computers with built in raster graphics displays popularized the use of bitmap graphics for array of points or also called as pixels on the screen. Once bitmap graphics became affordable, an explosion of easy-to-use and inexpensive graphics-based applications soon followed. Graphics-based interfaces allowed millions f news users to control simple, low cost application programs, Such as spreadsheets, word processors, and drawing programs. Graphics has its own hardcopy technologies, input technologies and also display technologies. Some of the hardcopy technologies are printers, pen plotters etc. Some of the display technologies such direct view storage tube, liquid crystal displays, plasma panels etc. Input technologies like keyboard, mouse, touch panel, tablets etc.

**1.2 Applications of Computer Graphics**

Computer graphics is used today in mainly different areas of industry, business, government, education, entertainment, and most recently, the home. The list of applications is enormous and is growing rapidly as computers with graphics capabilities become commodity products. Now let us see some of the applications.

* **Computer Aided Design**

Computer graphics is used in design process, particularly for engineering and architectural systems, but almost all products are now computer designed. Generally referred to as CAD, computer aided design methods are now routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles, and many other products.

* **Image Processing**

In computer graphics, a computer is used to create a picture. Image processing on the other hand, applies techniques to modify or interpret existing pictures, such as photographs and TV scans. Two principal applications of image processing are improving picture quality and machine perception of visual information, as used in robotics.

* **Education and Training**

Computer generated models of physical, financial and economic systems are often used as educational aids. Models of physical systems, physiological systems, population trends, or equipment, can help trainees to understand the operation of the system.

* **Entertainment**

Computer graphics methods are now commonly used in making motion pictures, music videos, and television shows.

* **Computer Simulation**

A computer simulation, a computer model or a computational model is a computer program, or network of computers, that attempts to simulate an abstract model of particular system.

* **Scientific Visualization**

It is a branch of science, concerned with the visualization of 3D phenomena, such as architectural, meteorological, medical, biological systems.

* **Virtual Reality**

It is a technology which allows a user to interact with a computer-simulated environment. The simulated environment can be similar to the real world.

* **Digital Art**

Digital Art most commonly refers to art created on a computer in digital form.

* **Web Design**

It is the skill of designing presentations of content usually hypertext or hypermedia that is delivered to an end-user through the World Wide Web, by way of a web browser.

**1.3.1 Common Functions of OpenGL**

|  |  |
| --- | --- |
| glBegin, glEnd | The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives. |
| GlClear | The glClear function clears buffers to preset values. |
| GlColor | These functions set the current color |
| GlFlush | The glFlush function forces execution of OpenGL functions in finite time. |
| GlLoadIdentity | The glLoadIdentity function replaces the current matrix with the identity matrix. |
| GlMatrixMode | The glMatrixMode function specifies which matrix is the current matrix. |
| GlVertex | These functions specify a vertex. |
| gluOrtho2D | The gluOrtho2D function defines a 2-D orthographic projection matrix. |

**1.3.2 Features of OpenGL**

OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use.

OpenGL is an API and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation.

OpenGL is a collection of several hundred functions that provide access to all of the features that your graphics hardware has to offer. Internally it acts like a state machine-a collection of states that tell OpenGL what to do. Using the API you can set various aspects of this state machine, including current color, blending, lighting effect, etc.

Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

**1.4 Advantages of OpenGL**

* It can be user-friendly and speed up the user's work.
* It can be more attractive for non-technical people.
* In general, it looks more professional (but this does not mean it is always the best solution).
* OpenGL is a cross-platform graphics API, which means that the same code can be used on multiple operating system types with minimal changes.
* OpenGL runs on every computer with graphics output capability and requires no extra downloads.

**1.5 Drawbacks of OpenGL**

* When it is not properly built, it can be very difficult to work with.
* It generally requires more resources than a non-graphical one.
* It might require the installation of additional software e.g., the “runtime Environment” in the case of java.
* OpenGL implementations can vary a lot, even certified ones.
* It has traditionally used a confusing state select model where you bind objects or bind state, which changes which state other functions change. Too bad the old functions are still available.
* OpenGL which leads to deprecation. For each new OpenGL version, the number of API functions explodes.

1.6 3D Shape Testing

In this project we are mainly testing the 3D shapes basically we are using six shapes

* Sphere
* Cube
* Cone
* Torus
* Icosahedron
* Teapot

OBJECTIVE: All these six shapes are rotating in an clockwise. By using left button of the mouse the speed can be increased.

If we click the right mouse button the shapes can be changed

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 Survey on the ship**

The computer graphics (CG) mini project called “Titanic” is a simple project built using OpenGL, where a ship will float in the sea. While the ship traveling in the sea in the middle an obstacle (iceberg) will be placed and the ship will hit the obstacle and the ship will sink into the sea.

The objects designed in this project are a ship and an obstacle (iceberg). A ship will be traveling in the sea when it reaches some extent an obstacle will appear in the way of the ship and the ship will hit the obstacle and the ship will sink soon into the sea. The controls will be specified and these controls will be provided from the keyboard.

To move the ship into the sea the control will be on the keyboard by pressing any key the ship will start to move in the sea and shows the beautiful voage in the sunny morning.the obstacle will appear automatically after some time in the way of the ship and the ship will hit the obstacle at night and sink in the sea soon.

**2.2 OpenGL**

The OpenGL specification describes an abstract [API](https://en.wikipedia.org/wiki/Application_programming_interface) for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely [in hardware](https://en.wikipedia.org/wiki/Hardware_acceleration).

The API is defined as a set of [functions](https://en.wikipedia.org/wiki/Subroutine) which may be called by the client program, alongside a set of [named integer constants](https://en.wikipedia.org/wiki/Enumerated_type) (for example, the constant GL\_TEXTURE\_2D, which corresponds to the [decimal](https://en.wikipedia.org/wiki/Decimal) number 3553). Although the function definitions are superficially similar to those of the programming language [C](https://en.wikipedia.org/wiki/C_(programming_language)), they are language-independent. As such, OpenGL has many [language bindings](https://en.wikipedia.org/wiki/Language_binding), some of the most noteworthy being the [JavaScript](https://en.wikipedia.org/wiki/JavaScript) binding [WebGL](https://en.wikipedia.org/wiki/WebGL) (API, based on [OpenGL ES 2.0](https://en.wikipedia.org/wiki/OpenGL_ES_2.0), for 3D rendering from within a [web browser](https://en.wikipedia.org/wiki/Web_browser)); the C bindings [WGL](https://en.wikipedia.org/wiki/WGL_(software)), [GLX](https://en.wikipedia.org/wiki/GLX) and [CGL](https://en.wikipedia.org/wiki/Core_OpenGL); the C binding provided by [iOS](https://en.wikipedia.org/wiki/IOS); and the [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) and C bindings provided by [Android](https://en.wikipedia.org/wiki/Android_(operating_system)).

In addition to being language-independent, OpenGL is also cross-platform. The specification says nothing on the subject of obtaining, and managing an OpenGL context, leaving this as a detail of the underlying [windowing system](https://en.wikipedia.org/wiki/Windowing_system). For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing

Another recent development has been the adoption of general purpose GPU (GPGPU) libraries, including nVidia's CUDA and Khorana’s OpenGL. These libraries implement dialects of C with added data parallelism features, allowing the GPU to be used for general computation without having to work within the graphics-oriented framework of OpenGL. However, these GPGPU frameworks don't replace OpenGL; since their primary purpose is not graphics programming, they only provide access to a GPU's computation units, ignoring its graphics-specific hardware. They can, however, act as accessories to OpenGL.We'll be focusing on using OpenGL for graphics tasks.  In addition to OpenGL, we'll be using other helper libraries: GLU (OpenGL Utility) contains several routines that use lower level OpenGL commands to perform such tasks as setting up matrices for specific viewing specifications and projection. GLUT (OpenGL Utility Toolkit), which provides a cross-platform interface between the window system and OpenGL. For a fallback, Mesa provides an open-source, cross-platform software OpenGL implementation that works on Windows and almost all UNIX platforms. Mesa is also the most common OpenGL implementation on Linux, where it also works with the X server to interface OpenGL with graphics hardware using "direct rendering interface" (DRI) drivers.

To install GLUT look for the binary packages on their respective sites. MacOS X comes with GLUT preinstalled. Most Linux distributions have GLUT available through their package system, though for GLUT, you may need to enable your distribution's optional "non-free" package repositories, since its license is not technically open.We use GLUT library for interacting with any screen windowing systems.

**CHAPTER 3**

**REQUIREMENT ANALYSIS**

**3.1 Domain Understanding**

The main objective is to develop a suitable OpenGL graphics package to implement basic computer graphics skills. The aim of the project is to shoot the block using arrow before the arrow ends using OpenGL.

**3.2 Classification of Requirements**

**3.2.1 User Requirements**

Program to demonstrate the shooting the block using arrows, in which designing and developing this project interesting and a good learning

**3.2.1.1 Requirement Collection**

Some of the primitives that are implemented in our project are referred by using the books and websites mentioned in bibliography and others by using built in functions of OpenGL.

**3.2.1.2 User Defined Functions**

* rock()
* display1()
* display2()
* display3()
* ship()
* water()
* sun()
* moon()
* cloud()
* DrawCircle(float cx, float cy, float r, int num\_segments)
* print()

**.2.2 System Requirements**

Here we are using header files namely <GL/glut.h>,functions in the OpenGL in windows are stored in a library usually referred as GL.GLUT uses only GL functions and also contains code for creating objects and simplifying, viewing, rather than using different library for each system here we are using readily available library called the OPENGL UTILITY TOOL KIT(GLUT).

**3.2.2.1 Software Requirements**

* + - Operating system: ubuntu
    - Programming language-C
    - OpenGL

**3.2.2.2 Hardware Requirements**

* Processor : Pentium processor
* Memory: 32MB RAM.
* Hard Disk : 40GB Hard disk
* Interface Device : Keyboard, Monitor of Resolution 1024x728

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Description of Implementation Modules**

In this project we have created a Titanic “OpenGL” functional API. We have taken the help of built in functions present in the header file. To provide functionality to our project we have written sub functions. These functions provide us the efficient way to design the project. In this chapter we are describing the functionality of our project using these functions.

**4.2 List of Implementation Functions**

**void rock():**

This function displays a static rock till the ships hits and breaks.

**void display1():**

This function displays a happy morning voage of Titanic ship.

**void display2():**

This function displays a scence where Titanic ship hits the iceberg.

**void display3():**

This function displays the sinking of ship.

**void ship():**

This function has the polygon function which draws the ship and translates.

**void water():**

This function has the polygon function which draws the water.

**void sun():**

This function has the polygon function which draws the sun.

**void moon():**

This function has the polygon function which draws the moon.

**void cloud():**

This function has the polygon function which draws the clouds.

**void DrawCircle(float cx, float cy, float r, int num\_segments):**

This function has the polygon function which is called by cloud to draw clouds.

**void print():**

This function is used to display the text on the screen after ship sinks.

**4.3 Description of the functions**

**main ():**

The execution of the program starts from the main ().

**glutInit ():**

Initialises GLUT. The argument from main are passed in and can be used by the application.

**glutInitWindowSize():**

Specifies the initial height and width of the window in pixels.

**glutCreateWindow():**

Creates the window on the display. The string can be used to label the window.

**glutDisplayFunc():**

Registers the display function that is executed when the window needs to redrawn.

**glutMainLoop():**

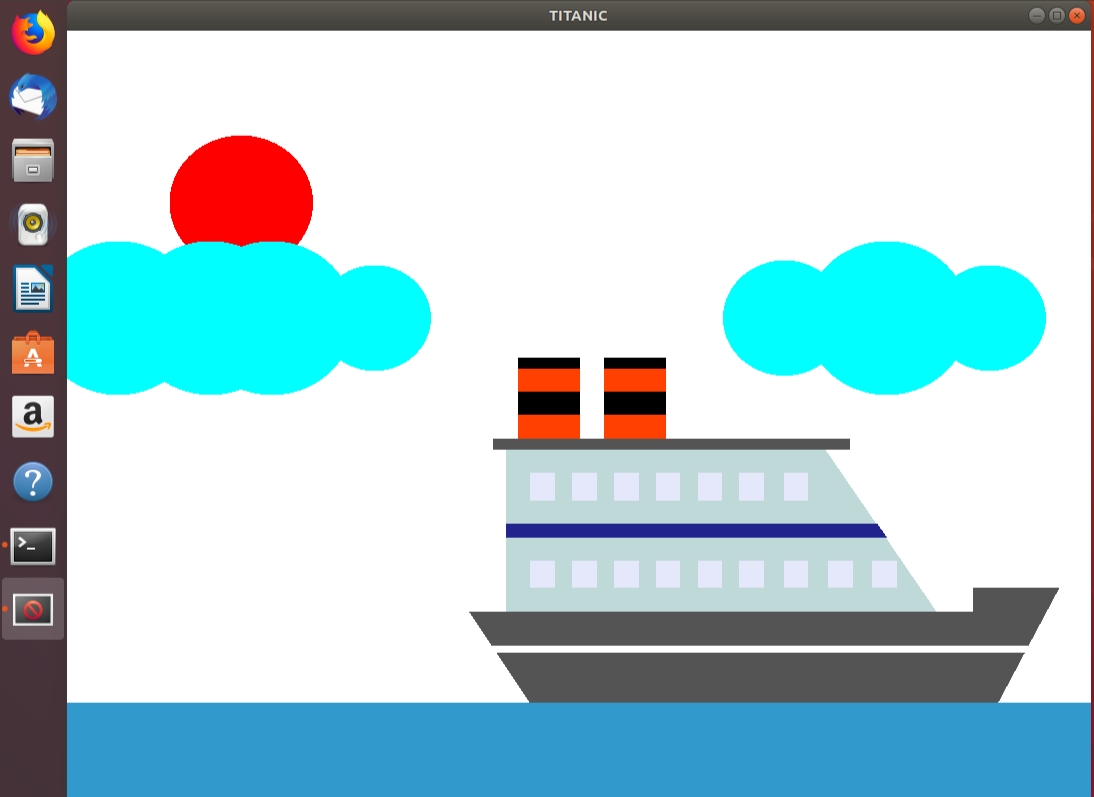
Causes the program to enter an event processing loop.

**myinit():**

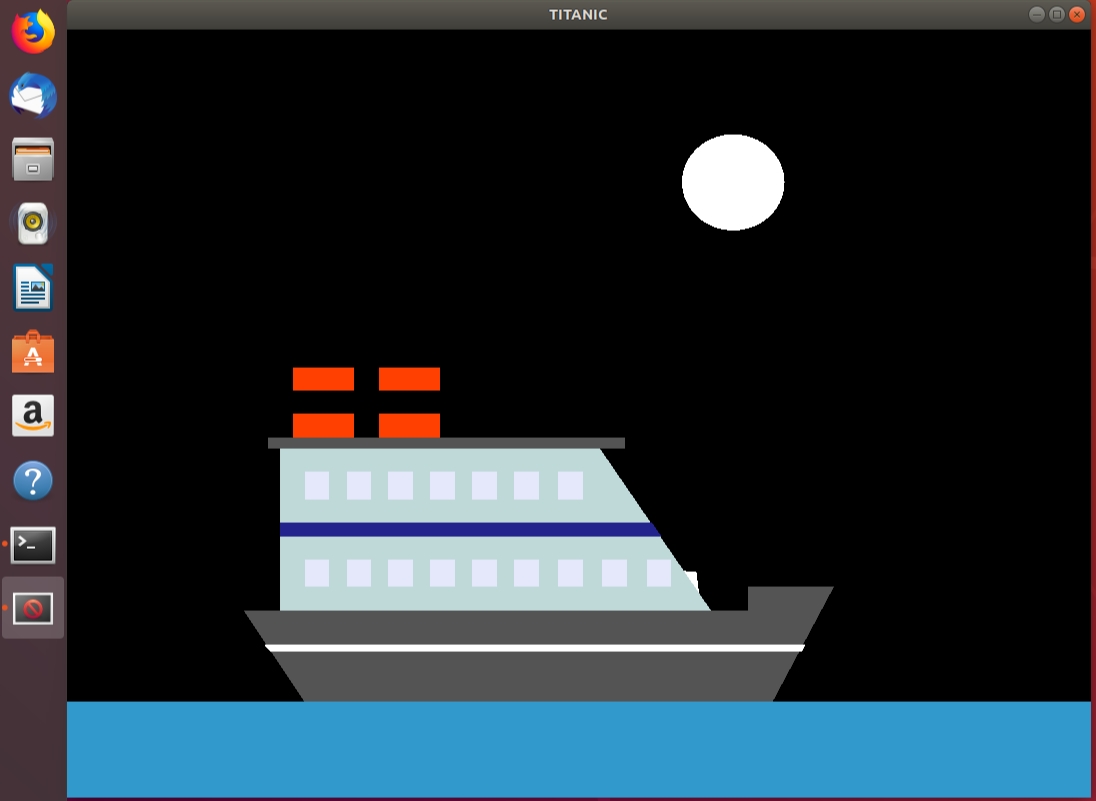
This function is defined to initialize the window parameters.

**CHAPTER 5**

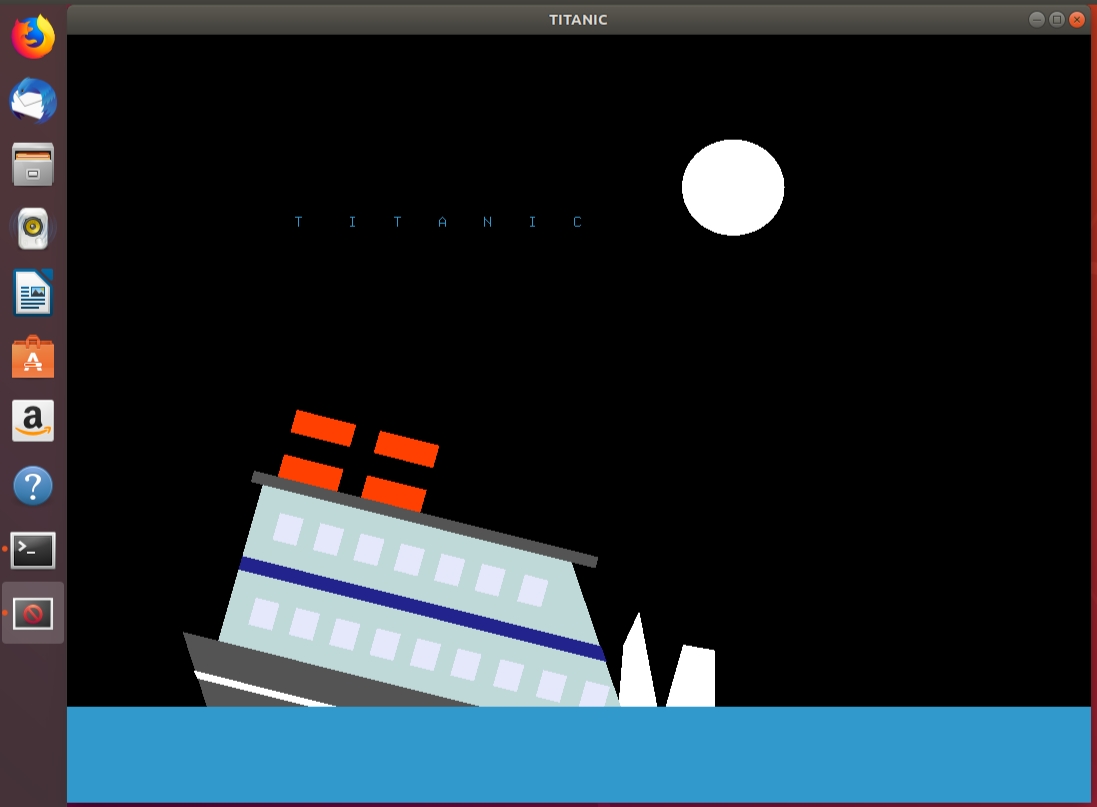
**SNAPSHOTS**



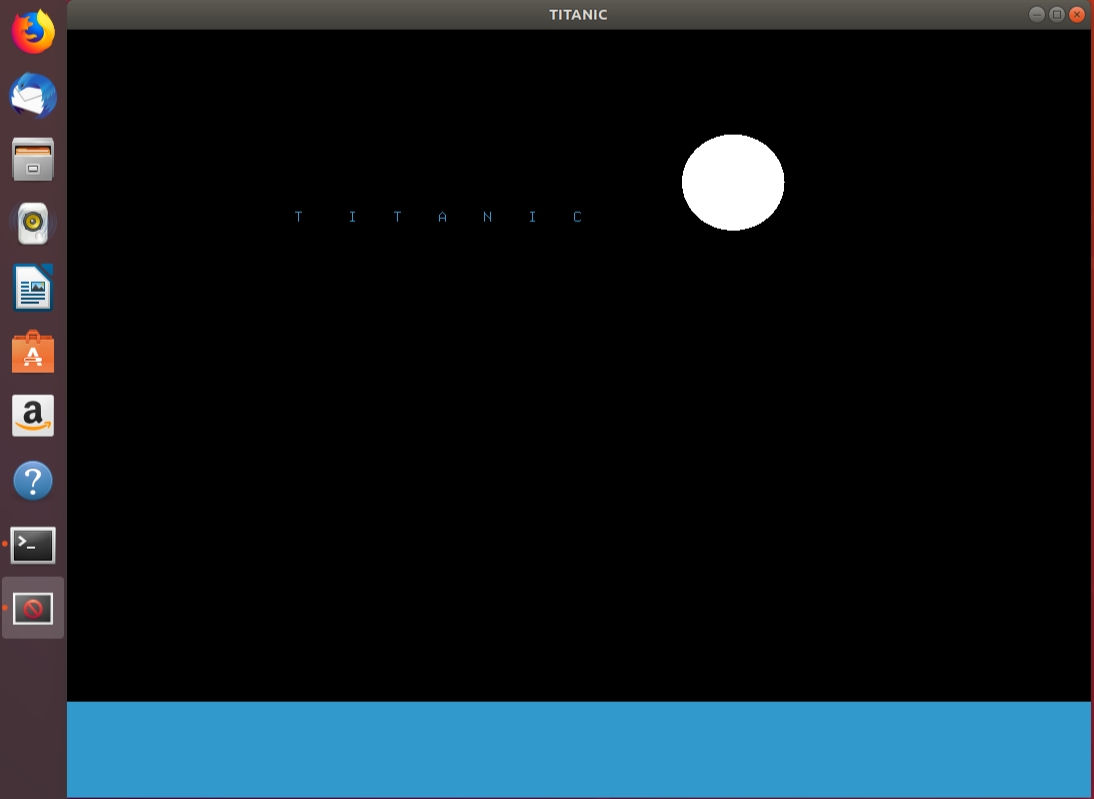
**Fig 5.1 DISPLAY 1 (Happy Voage)**

****

**Fig 5.2 DISPLAY 2 (Ship Hits Iceberg)**

****

**Fig 5.3 Sinking Of The Ship**

****

**Fig 5.4 DISPLAY 3 (The End of Titanic)**

**CONCLUSION**

These kinds of projects are simple to implement by implementing these simple projects you will get a lot of knowledge about designing the objects so that you can implement some more objects and effects to this simple project. It also helped us to understand various inbuilt functions of OpenGL and allowed to implement my own. The keyboard interaction has helped me to implement this project in more easy way and OpenGL methodology was employed, I obtain a good expressive in OpenGL software development.

We conclude the project ‘Titanic’ successfully truest of our senses and to best of my ability.

**FUTURE ENHANCEMENTS**

* Add Sounds and Music according to the scene.
* Create stars and add movements for the stars while the ship is traveling in the sea.

**REFERENCES**

1. S.M. Dorman, “Video and computer games: effect on children and implications for health education,” *Journal of School Health*, vol. 67, no. 4, pp. 133–138, 1997.
2. Interactive Computer Graphics A Top-Down Approach with OpenGL- Edward Angel, 6th Edition, Addison-Wesley, 2008.
3. <https://www.gamedeveloperstudio.com/> - For the objects used – Tank Alien ships etc.
4. <https://opengameart.org/> - For the art used in the project – Background, Textures etc.
5. <https://lazyfoo.net/tutorials/OpenGL/> - Tutorials.

1. For textbooks - A.V. Oppenheim and R.W. Schafer, Digital Signal Processing, Englewood, N.J., Prentice Hall, 3 Edition, 1975.
2. For papers - David, Insulation design to combat pollution problem, Proc of IEEE, PAS, Vol 71, Aug 1981, pp 1901-1907.

.

**APPENDIX**

#include<stdio.h>

#include<GL/glut.h>

#include<stdlib.h>

#include<time.h>

#include<math.h>

#include<string.h>

const float PI =3.141592653;

void rock();

void display1();

void display2();

void display3();

void ship();

void water();

void sun();

void moon();

void cloud();

void DrawCircle(float cx, float cy, float r, int num\_segments);

void print();

struct timespec jmp, jmp2;

GLint a=0,b=0,c=0,d=0,e=0,f=0,g=500,h=600,x=0,i=0;

GLfloat theta=0.0;

float xc=0.0,yc=0.0,x1c=5.0;

void update(int value)

{

a+=20.0;

glutPostRedisplay(); //which display must be displayed for second loop

glutTimerFunc(80,update,0); //200

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

display1();

if(a>950)

{

b+=20;

display2();

}

if(b==180)

{

jmp.tv\_sec = 2; //handles seconds

jmp.tv\_nsec = 500000000L; //handles nanoseconds

nanosleep(&jmp , &jmp2);

}

if(b>250)

{

c+=10;

display3();

}

glFlush(); //pushes all the executable commands from buffer to execute

glutSwapBuffers();

}

void display1()

{

sun();

cloud();

glPushMatrix(); // Set current matrix on the stack

glTranslated(a,75,0.0); //glTranslatef(someX=a, someY=75, someZ=0); transformation 1

ship(); //build a ship DrawObject(ONE);

glPopMatrix(); // Pop the old matrix without the transformations.

water(); //u need water to float

}

void display2()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0f, 0.0f, 0.0f, 0.0f); //ext

moon();

rock();

glPushMatrix();

glTranslated(b,75,0.0);

ship();

glPopMatrix();

water();

}

void display3()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

moon();

glPopMatrix();

x-=1;

rock();

glPushMatrix();

glTranslated(c,100+x\*5,0.0);

glRotated(-15,0,0,1);

ship();

glPopMatrix();

water();

print();

}

void sun()

{

float angle;

int xp=170;

int y=620;

int radius=70;

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

for (int i = 0;i < 100;i++)

{

angle = i \* 2 \* (PI / 100);

glVertex2f(xp + (cos(angle) \* radius), y + (sin(angle) \* radius)); //-400

}

glEnd();

//ext ends

}

void moon()

{

float angle;

int xp=650;

int y=640;

int radius=50;

glColor3f(217.0,217.0,217.0);

glBegin(GL\_POLYGON);

for (int i = 0;i < 100;i++)

{

angle = i \* 2 \* (PI / 100);

glVertex2f(xp + (cos(angle) \* radius), y + (sin(angle) \* radius)); //-400

}

glEnd();

//ext ends

}

void water()

{

glColor3f(0.196078,0.6,0.8);

glBegin(GL\_POLYGON);

glVertex2f(0,0);

glVertex2f(1000,0);

glVertex2f(1000,100);

glVertex2f(0,100);

glEnd();

}

void cloud()

{

glColor3f(0.0, 255.0, 255.0);

DrawCircle(30, 500, 60, 2000);//1

DrawCircle(50,500, 80, 2000);//2

DrawCircle(100, 500, 55, 2000);//3

DrawCircle(130, 500, 60, 2000);//1

DrawCircle(140,500, 80, 2000);//2

DrawCircle(150,500, 55, 2000);//3

DrawCircle(170, 500, 60, 2000);//1

DrawCircle(200,500, 80, 2000);//2

DrawCircle(300, 500, 55, 2000);//3

DrawCircle(700, 500, 60, 2000);//1

DrawCircle(800,500, 80, 2000);//2

DrawCircle(900, 500, 55, 2000);//3

}

void DrawCircle(float cx, float cy, float r, int num\_segments){

float x;

float y;

glBegin(GL\_TRIANGLE\_FAN);

for (int i = 0; i < num\_segments; i++)

{

float theta = (2.0f \* 3.1415926f \*i)/(num\_segments);//get the current angle

x = r \* cosf(theta);//calculate the x component

y = r \* sinf(theta);//calculate the y component

glVertex2f(x + cx, y + cy);//output vertex

}

glEnd();

}

void print()

{

int x=90;

int y=600;

int z=0;

char \*string=" T I T A N I C ";

//set the position of the text in the window using the x and y coordinates

glRasterPos2f(x,y);

//get the length of the string to display

int len = (int) strlen(string);

//loop to display character by character

for (int i = 0; i<len; i++)

{

glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15,string[i]);

}

}

void ship()

{

glScaled(24,24,0);

glColor3f(0.329412,0.329412,0.329412); //base

glBegin(GL\_POLYGON);

glVertex2f(0.5,5.0);

glVertex2f(3,1);

glVertex2f(22,1);

glVertex2f(24.0,5.0);

glEnd();

glColor3f(1.0,1.0,1.0); //ring

glBegin(GL\_POLYGON);

glVertex2f(1.35,3.5);

glVertex2f(1.6,3.2);

glVertex2f(23.2,3.2);

glVertex2f(23.35,3.5);

glEnd();

glColor3f(0.329412,0.329412,0.329412); //base

glBegin(GL\_POLYGON);

glVertex2f(21.0,5.0);

glVertex2f(21.0,6.0);

glVertex2f(24.5,6.0);

glVertex2f(24.0,5.0);

glEnd();

glColor3f(0.74902,0.847059,0.847059); //top-mid

glBegin(GL\_POLYGON);

glVertex2f(2.0,5.0);

glVertex2f(2.0,12.0);

glVertex2f(15.0,12.0);

glVertex2f(19.5,5.0);

glEnd();

glColor3f(0.137255,0.137255,0.556863); //ring

glBegin(GL\_POLYGON);

glVertex2f(2.0,8.2);

glVertex2f(2.0,8.8);

glVertex2f(17.1,8.8);

glVertex2f(17.5,8.2);

glEnd();

glColor3f(0.90,0.91,0.98); //window

glBegin(GL\_POLYGON);

glVertex2f(3.0,6.0);

glVertex2f(3.0,7.2);

glVertex2f(4.0,7.2);

glVertex2f(4.0,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(4.7,6.0);

glVertex2f(4.7,7.2);

glVertex2f(5.7,7.2);

glVertex2f(5.7,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(8.1,6.0);

glVertex2f(8.1,7.2);

glVertex2f(9.1,7.2);

glVertex2f(9.1,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(9.8,6.0);

glVertex2f(9.8,7.2);

glVertex2f(10.8,7.2);

glVertex2f(10.8,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(6.4,6.0);

glVertex2f(6.4,7.2);

glVertex2f(7.4,7.2);

glVertex2f(7.4,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(11.5,6.0);

glVertex2f(11.5,7.2);

glVertex2f(12.5,7.2);

glVertex2f(12.5,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(13.3,6.0);

glVertex2f(13.3,7.2);

glVertex2f(14.3,7.2);

glVertex2f(14.3,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(15.1,6.0);

glVertex2f(15.1,7.2);

glVertex2f(16.1,7.2);

glVertex2f(16.1,6.0);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(16.9,6.0);

glVertex2f(16.9,7.2);

glVertex2f(17.9,7.2);

glVertex2f(17.9,6.0);

glEnd();

glColor3f(0.90,0.91,0.98); //window

glBegin(GL\_POLYGON);

glVertex2f(3.0,9.8);

glVertex2f(3.0,11.0);

glVertex2f(4.0,11.0);

glVertex2f(4.0,9.8);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(4.7,9.8);

glVertex2f(4.7,11.0);

glVertex2f(5.7,11.0);

glVertex2f(5.7,9.8);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(8.1,9.8);

glVertex2f(8.1,11.0);

glVertex2f(9.1,11.0);

glVertex2f(9.1,9.8);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(9.8,9.8);

glVertex2f(9.8,11.0);

glVertex2f(10.8,11.0);

glVertex2f(10.8,9.8);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(6.4,9.8);

glVertex2f(6.4,11.0);

glVertex2f(7.4,11.0);

glVertex2f(7.4,9.8);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(11.5,9.8);

glVertex2f(11.5,11.0);

glVertex2f(12.5,11.0);

glVertex2f(12.5,9.8);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(13.3,9.8);

glVertex2f(13.3,11.0);

glVertex2f(14.3,11.0);

glVertex2f(14.3,9.8);

glEnd();

glColor3f(0.329412,0.329412,0.329412); //top-cover

glBegin(GL\_POLYGON);

glVertex2f(1.5,12.0);

glVertex2f(1.5,12.5);

glVertex2f(16.0,12.5);

glVertex2f(16.0,12.0);

glEnd();

glColor3f(0.0,0.0,0.0); //chim

glBegin(GL\_POLYGON);

glVertex2f(2.5,12.5);

glVertex2f(2.5,16.0);

glVertex2f(5.0,16.0);

glVertex2f(5.0,12.5);

glEnd();

glColor3f(1.0,0.25,0.0); //ring

glBegin(GL\_POLYGON);

glVertex2f(2.5,12.5);

glVertex2f(2.5,13.5);

glVertex2f(5.0,13.5);

glVertex2f(5.0,12.5);

glEnd();

glColor3f(1.0,0.25,0.0); //ring

glBegin(GL\_POLYGON);

glVertex2f(2.5,14.5);

glVertex2f(2.5,15.5);

glVertex2f(5.0,15.5);

glVertex2f(5.0,14.5);

glEnd();

glColor3f(0.0,0.0,0.0); //chim

glBegin(GL\_POLYGON);

glVertex2f(6.0,12.5);

glVertex2f(6.0,16.0);

glVertex2f(8.5,16.0);

glVertex2f(8.5,12.5);

glEnd();

glColor3f(1.0,0.25,0.0); //ring

glBegin(GL\_POLYGON);

glVertex2f(6.0,14.5);

glVertex2f(6.0,15.5);

glVertex2f(8.5,15.5);

glVertex2f(8.5,14.5);

glEnd();

glColor3f(1.0,0.25,0.0); //ring

glBegin(GL\_POLYGON);

glVertex2f(6.0,12.5);

glVertex2f(6.0,13.5);

glVertex2f(8.5,13.5);

glVertex2f(8.5,12.5);

glEnd();

}

void rock()

{

//ext

//ext

glPushMatrix();

glTranslated(450,50,0.0);

glScaled(10,10,0);

glColor3f(255,255,255); //glColor3f(0.36,0.25,0.20);

if(c>0) //break the rock as c is inc in display 3

{

glPushMatrix();

glTranslated(0,x,0);

glPushMatrix();

glTranslated(7,2,0.0);

glRotated(-x,0,0,1);

glTranslated(-7,-2,0.0);

glBegin(GL\_POLYGON);

glVertex2f(8.1,1);

glVertex2f(7.89,1.23);

glVertex2f(11.26,18);

glVertex2f(13.23,21.33);

glVertex2f(14.23,2.53);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(12,3,0.0);

glRotated(x,0,0,1);

glTranslated(-12,-3,0.0);

glBegin(GL\_POLYGON);

glVertex2f(11.1,3.5);

glVertex2f(13.2,18.7);

glVertex2f(16.4,18.5);

glVertex2f(18.12,3.65);

glVertex2f(19.21,1.25);

glEnd();

glPopMatrix();

glPopMatrix();

}

else //static rock

{

glBegin(GL\_POLYGON);

glVertex2f(8.1,1);

glVertex2f(7.89,1.23);

glVertex2f(11.26,18);

glVertex2f(13.23,21.33);

glVertex2f(14.23,2.53);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(11.1,3.5);

glVertex2f(13.2,18.7);

glVertex2f(16.4,18.5);

glVertex2f(18.12,3.65);

glVertex2f(19.21,1.25);

glEnd();

}

glPopMatrix();

}

void myinit()

{

glClearColor(1.0f,1.0f,1.0f,1.0f);

glColor3f(1.0,0.0,0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,999.0,0.0,799.0);

}

int main(int argc, char\* argv[])

{

int chs;

printf("Type Any Key and Hit Enter\n");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

scanf("%d",&chs);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(1024.0,768.0);

glutInitWindowPosition(0,0);

glutCreateWindow("TITANIC");

glutDisplayFunc(display);

myinit();

glutTimerFunc(100,update,0);

glutMainLoop();

return 0;

}