**CHAPTER 1**

**INTRODUCTION**

* 1. **Computer Graphics**

Computer graphics are graphics created using computer and more generally the representation and manipulation of pictorial data by a computer.

The development of computer graphics has made computers. The development of computer graphics has made computer easier to interact with and better for understanding and interpreting many types of data. Development in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry.

The term computer graphics includes almost everything on computers that is not in text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

* The representation and manipulation of pictorial data by computer.
* The various technologies used and manipulation such pictorial data.
* The images so produced.

There are two types of Computer Graphics – raster and vector. The raster graphics is composed of pixels and vector graphics is composed of drawing paths. It is important to understand the difference between these two types before you choose the graphics format to save the barcode image.

**1. 2 Introduction to OpenGL**

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL was developed by Silicon Graphics Inc. from 1991 and released in January 1992.

Several libraries are built on top of OpenGL to provide features not available in OpenGL itself:

* GLU
* GLUT
* GL

OpenGL or Open Graphics Library 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. It provides the actual drawing tools through a collection of functions that are called within an application. It is easy to install and learn, and its longevity as a standard API is being nurtured and overseen by the OpenGL Architecture Review Board (ARB), an industry consortium responsible for guiding its evolution. It is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation and video games.

* 1. **Rocket Mission**

This mini project under Computer Graphics & Visualization Laboratory is an implementation of a game called rocket mission using the OpenGL Graphics Library and GLUT Toolkit.

The rocket is launched from the ground to space. The ground is filled with green color, it consist of a building, an antenna and a road. With the interaction of the keyboard we can change the position of the rocket, when it is in space. As it reaches space the moon, stars and asteroids appear.

**CHAPTER 2**

**SYSTEM REQUIREMENT ANALYSIS**

**2.1 Functional Requirements**

Functional requirements defines the internal working of the software that is, the calculations, technical detail, data manipulation and processing and other specific functionality that show how the cases are to satisfied and how they are supported by non-functional requirement, which impose constraints on the design or the implementation.

The following must be taken care of,

* The ability to perform correct operation when the corresponding keys are pressed.
* The ability to display the menu when the right mouse button is clicked.
* When the corresponding menu is selected the corresponding option should be performed.

**2.2 Non-Functional Requirements**

Non-Functional Requirements are requirements which specify criteria that can be used to judge the operations of the system, rather than specific behaviour. This should be contrasted with functional requirement that specify behaviour or functions. Typical non-functional requirements are reliability and scalability. Non-functional requirement are “constraints”, “quality attribute” and quality of service requirement.

**2.3 Software Requirements**

* **Operating System:** Microsoft Windows series
* **Language:** C++
* **Tool:** MicrosoftVisual Studio
* **Library:** OpenGL

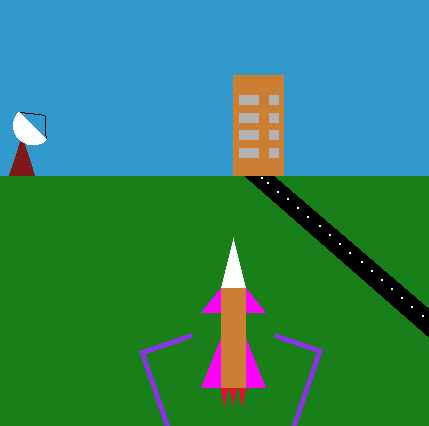
**2.4 Hardware Requirements**

* 512MB (or higher) RAM
* A minimum of 40 GB hard disk space
* Keyboard
* Colour monitor
* Mouse
* Processor Pentium (or higher) series

**CHAPTER 3**

**SYSTEM DESIGN**

Computers have become a powerful medium for a rapid economic production of picture. There is virtually no area in which graphical displays cannot be used to some advantage. Graphics provides a natural means of communicating with a computer that they have become wide spread. The field in which computer graphics find their uses are many. Some of them User Interface, Computer Aided Design Office Automation, Desktop publishing, Plotting of mathematical, Scientific and Industrial data, Simulation, Art, Presentation, Cartography, to name a few. Here we have tried to incorporate and present the project rocket mission.



**Figure 3.1: Rocket Mission snapshot**

In the beginning I tried to analyse topics that can be designed using simple OpenGL functions. In this project I have created a scene of launching of the rocket and shown how the rocket reaches space. Later it reaches the intermediate state where the dark blue sky occurs, then finally the rocket reaches space. As it approaches the outer-space the stars and moon appear and once the rocket is in outer-space it has to avoid the asteroids.

The basic features of rocket mission :-

* The score, speed and level are displayed once the rocket is in outer space. The score increase as the rocket avoids the asteroids.
* The rocket is moved in left and right directions with the respective arrow keys on the keyboard. The top and bottom arrow keys increase and decrease the speed of the rocket.
* When the rocket collides with an asteroid the screen changes and the message game over appears on the screen.
* By pressing the spacebar key, the game restarts, and pressing the esc key on the keyboard closes the game i.e. the program.

**CHAPTER 4**

**IMPLEMENTATION AND CODING**

**4.1 Implementation**

I have used several user defined functions and few built in functions. Apart from this I have used few of the computer graphics concepts which make the project an excellent implementation of the ideas in graphics.

**4.1.1 User defined functions**

**void display1()**

This function is used to start the launching, intermediate and final stage of the rocket.

**void stars()**

This function contains the code stars that appear on the screen as the rocket reaches the final stage.

**void moon(float radius)**

This function has the code for the moon which appears on the screen in later part of the final stage.

**void static\_rocket()**

In this function the background, buildings, road and antenna are displayed along with this the rocket with its stands.

**void rocket\_to\_cam\_pos()**

In this function the rocket move from the ground towards space with the background changing from dark blue to black.

**void rocket\_in\_motion()**

This the function that contains the main sequence of the game, where the rocket has to evade the incoming asteroids.

**4.1.2 Built-in functions**

**glutInit():**

It is used to initialize the GLUT library.

**glutInitWindowSize():**

Specifies the size in pixels of your computer.

**glutInitWindowPosition():**

Specifies the screen location for the upper left corner of your window.

**glutInitDisplayMode():**

Specify whether to use an RGBA or color index color model.

**glutCreateWindow():**

Creates window with an OpenGL context. It returns a unique identifier for the window.

**glutDisplayFunc():**

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

**glutKeyboardFunc():**

Allow you to link a keyboard key with a routine.

**glPushMatrix(), glPopMatrix() :**

Pushes to and pops from the matrix stacks corresponding to the current matrix mode.

**glTranslate():**

Alerts the current matrix by a displacement of (x, y, z).

**glMatrixMode():**

This function specifies which matrix will be affected by subsequent transformations.

Mode can be GL\_MODELVIEW, GL\_PROJECTIONVIEW, GL\_TEXTURE.

**glFlush():**

Thisfunction forces any buffered OpenGL commands to execute.

**glBegin():**

This initiates a new primitive of type mode and starts the collection of vertices.

Values of mode includes GL\_POINTS, GL\_LINES AND GL\_POLYGON.

**glSwapBuffers():**

When one buffer gets filled the next buffer will get ready to accept the data.

**glutPostRediplay():**

This will help in the continuous display of output.

**glEnd():**

It terminates a list of vertices.

**glutMainLoop():**

Causes the program to enter an event processing loop, it should be the last statement in the main.

**4.2 CODING**

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<glut.h>

//Game Speed

int FPS = 50;

//Game Track

int start = 0;

int gv = 0;

int level = 0;

//Track Score

int score = 0;

//For astr Left / RIGHT

int lrIndex = 0; //main astr

int astr1 = 0; //astr Coming

int lrIndex1 = 0; //red astr

int astr2 = +35;

int lrIndex2 = 0;//violet astr

int astr3 = +70;

int lrIndex3 = 0;

//For Display TEXT

const int font1 = (int)GLUT\_BITMAP\_TIMES\_ROMAN\_24;

const int font2 = (int)GLUT\_BITMAP\_HELVETICA\_18;

const int font3 = (int)GLUT\_BITMAP\_8\_BY\_13;

const float DEG2RAD = 3.14159 / 180;

void display1();

void stars();

int p;

void stars1();

void static\_rocket();

void rocket\_to\_cam\_pos();

void rocket\_in\_motion();

void moon(float radius);

float i, j,

count = 0, /\*used to count delay in launching\*/

count1 = 0, /\*used to count blue sky delay\*/

count3 = 0, /\*used to count black sky delay\*/

flag = 0, flag1 = 0, t = 0, f = 0, flag3 = 0,

t1 = 0 /\* used to translate rocket initially\*/

;

void display1()

{

count1++;

//if(count1==250)

// flag=1;

if (flag == 0)

static\_rocket();

else if ((count1 == 751) || (count1 == 752))

rocket\_to\_cam\_pos();

else rocket\_in\_motion();

}

void rocket\_in\_motion()//rocket in outer space

{

count++;

for (i = 195; i <= 200; i++)

{

if (count >= 5)

{

glClearColor(0.0, 0.0, 0.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

if (flag1 == 0)

{

stars();

flag1 = 1;

}

else

{

stars1();

flag1 = 0;

}}

else

{

glClearColor(0.0, 0, 0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

}

if (count >= 100)

moon(20.0);

glColor3f(0.8, 1.498039, 0.0);

glBegin(GL\_POLYGON);//core of the rocket

glVertex2f(lrIndex + 237.5, -100.0 + i);

glVertex2f(lrIndex + 262.5, -100.0 + i);

glVertex2f(lrIndex + 262.5, 10.0 + i);

glVertex2f(lrIndex + 237.5, 10.0 + i);

glEnd();

glColor3f(1.0, 1.0, 1.0);//bonnet

glBegin(GL\_POLYGON);//front

glVertex2f(lrIndex + 237.5, 10.0 + i);

glVertex2f(lrIndex + 262.5, 10.0 + i);

glVertex2f(lrIndex + 250, 60.0 + i);

glEnd();

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);//left\_side\_top fin

glVertex2f(lrIndex + 237.5, 10.0 + i);

glVertex2f(lrIndex + 217.5, -15.0 + i);

glVertex2f(lrIndex + 237.5, -15.0 + i);

glEnd();

glBegin(GL\_POLYGON);//left\_side\_bottom fin

glVertex2f(lrIndex + 237.5, -100.0 + i);

glVertex2f(lrIndex + 217.5, -100.0 + i);

glVertex2f(lrIndex + 237.5, -50.0 + i);

glEnd();

glBegin(GL\_POLYGON);//right\_side\_bottom fin

glVertex2f(lrIndex + 262.5, -100.0 + i);

glVertex2f(lrIndex + 282.5, -100.0 + i);

glVertex2f(lrIndex + 262.5, -50.0 + i);

glEnd();

glBegin(GL\_POLYGON);//right\_side\_top fin

glVertex2f(lrIndex + 262.5, 10.0 + i);

glVertex2f(lrIndex + 262.5, -15.0 + i);

glVertex2f(lrIndex + 282.5, -15.0 + i);

glEnd();

glColor3f(0.556863, 0.137255, 0.419608);

glBegin(GL\_POLYGON);//bottom\_1\_exhaust

glVertex2f(lrIndex + 237.5, -100.0 + i);

glVertex2f(lrIndex + 244.5, -100.0 + i);

glVertex2f(lrIndex + 241, -100.0 + i);

glEnd();

glBegin(GL\_POLYGON);//bottom\_2\_exhaust

glVertex2f(lrIndex + 246.5, -100.0 + i);

glVertex2f(lrIndex + 253.5, -100.0 + i);

glVertex2f(lrIndex + 249.5, -100.0 + i);

glEnd();

glBegin(GL\_POLYGON);//bottom\_3\_exhaust

glVertex2f(lrIndex + 262.5, -100.0 + i);

glVertex2f(lrIndex + 255.5, -100.0 + i);

glVertex2f(lrIndex + 258.5, -100.0 + i);

glEnd();

if ((p % 2) == 0)

glColor3f(1.0, 0.25, 0.0);

else

glColor3f(1.0, 0.816, 0.0);

glBegin(GL\_POLYGON);//outer fume

glVertex2f(lrIndex + 237.5, -100 + i);

glVertex2f(lrIndex + 234.16, -104.44 + i);

glVertex2f(lrIndex + 230.82, -107.78 + i);

glVertex2f(lrIndex + 227.48, -111.12 + i);

glVertex2f(lrIndex + 224.14, -114.46 + i);

glVertex2f(lrIndex + 220.8, -117.7 + i);

glVertex2f(lrIndex + 217.5, -120 + i);

glVertex2f(lrIndex + 221.56, -125 + i);

glVertex2f(lrIndex + 225.62, -130 + i);

glVertex2f(lrIndex + 229.68, -135 + i);

glVertex2f(lrIndex + 233.74, -140 + i);

glVertex2f(lrIndex + 237.8, -145 + i);

glVertex2f(lrIndex + 241.86, -150 + i);

glVertex2f(lrIndex + 245.92, -155 + i);

glVertex2f(lrIndex + 250, -160 + i);

glVertex2f(lrIndex + 254.06, -165 + i);

glVertex2f(lrIndex + 258.12, -150 + i);

glVertex2f(lrIndex + 262.18, -145 + i);

glVertex2f(lrIndex + 266.24, -140 + i);

glVertex2f(lrIndex + 270.3, -135 + i);

glVertex2f(lrIndex + 274.36, -130 + i);

glVertex2f(lrIndex + 278.42, -125 + i);

glVertex2f(lrIndex + 282.5, -120 + i);

glVertex2f(lrIndex + 278.5, -116 + i);

glVertex2f(lrIndex + 274.5, -112 + i);

glVertex2f(lrIndex + 270.5, -108 + i);

glVertex2f(lrIndex + 266.5, -104 + i);

glVertex2f(lrIndex + 262.5, -100 + i);//28 points

glEnd();

if ((p % 2) == 0)

glColor3f(1.0, 0.816, 0.0);

else

glColor3f(1.0, 0.25, 0.0);

glBegin(GL\_POLYGON);//inner fume

glVertex2f(lrIndex + 237.5, -100 + i);

glVertex2f(lrIndex + 236.5, -103.5 + i);

glVertex2f(lrIndex + 235.5, -105 + i);

glVertex2f(lrIndex + 234.5, -108.5 + i);

glVertex2f(lrIndex + 233.5, -110 + i);

glVertex2f(lrIndex + 232.5, -113.5 + i);

glVertex2f(lrIndex + 236, -115 + i);

glVertex2f(lrIndex + 239.5, -118.5 + i);

glVertex2f(lrIndex + 243, -120 + i);

glVertex2f(lrIndex + 246.5, -122.5 + i);

glVertex2f(lrIndex + 250, -125 + i);

glVertex2f(lrIndex + 253.5, -122.5 + i);

glVertex2f(lrIndex + 257, -120 + i);

glVertex2f(lrIndex + 260.5, -122.5 + i);

glVertex2f(lrIndex + 264, -125 + i);

glVertex2f(lrIndex + 267.5, -127.5 + i);

glVertex2f(lrIndex + 266.5, -130 + i);

glVertex2f(lrIndex + 265.5, -132.5 + i);

glVertex2f(lrIndex + 264.5, -135 + i);

glVertex2f(lrIndex + 263.5, -137.5 + i);

glVertex2f(lrIndex + 262.5, -140 + i);//21 points

glEnd();

p = p + 1;

//Score Board On The Running Game Screen

glColor3f(1.000, 0.000, 0.000);

glBegin(GL\_POLYGON);

glVertex2f(10, 90);

glVertex2f(110, 90);

glVertex2f(110, 90 - 40);

glVertex2f(10, 90 - 40);

glEnd();

//Asteroid

//Opposite Asteroid1

glColor3f(1.000, 0.000, 0.000);

glBegin(GL\_POLYGON);//Body of the asteroid

glVertex2f(lrIndex1 + 95, astr1 + 500);

glVertex2f(lrIndex1 + 105, astr1 + 510 - 9);

glVertex2f(lrIndex1 + 110, astr1 + 505 - 2);

glVertex2f(lrIndex1 + 99, astr1 + 495 - 7);

glVertex2f(lrIndex1 + 140, astr1 + 509 - 7);

glVertex2f(lrIndex1 + 142, astr1 + 514);

glVertex2f(lrIndex1 + 95 + 2, astr1 + 500 - 2);

glEnd();

astr1--;

if (astr1<-500) {

astr1 = 0;

lrIndex1 = lrIndex;

printf("%d", astr1);

}

//KIll check asteroid

if ((abs(lrIndex - lrIndex1)<8) && (astr1 + 500<10)) {

start = 0;

gv = 1;

}

//Opposite asteroid 2

glColor3f(0.294, 0.00, 0.510);

glBegin(GL\_POLYGON);//Body of the asteroid

glVertex2f(lrIndex2 + 400, astr2 + 399);

glVertex2f(lrIndex2 + 420, astr2 + 410 - 7);

glVertex2f(lrIndex2 + 430 - 2, astr2 + 430 - 2);

glVertex2f(lrIndex2 + 447 + 2, astr2 + 435 - 2);

glVertex2f(lrIndex2 + 450 - 2, astr2 + 399 - 2);

glVertex2f(lrIndex2 + 400 + 2, astr2 + 388 - 2);

glVertex2f(lrIndex2 + 410, astr2 + 390);

glEnd();

astr2--;

if (astr2<-500) {

astr2 = 0;

//Score

score++;

lrIndex2 = lrIndex;

}

//KIll check asteroid2

if ((abs(lrIndex - lrIndex2)<8) && (astr2 + 500<10)) {

start = 0;

gv = 1;

}

//Opposite asteroid 3

glColor3f(0.02390, 0.571, 0.0);

glBegin(GL\_POLYGON);//Body of the asteroid

glVertex2f(lrIndex3 + 270, astr3 + 500);

glVertex2f(lrIndex3 + 280, astr3 + 520 - 7);

glVertex2f(lrIndex3 + 330, astr3 + 500 - 9);

glVertex2f(lrIndex3 + 310 - 2, astr3 + 490 - 2);

glVertex2f(lrIndex3 + 295 + 2, astr3 + 500 - 2);

glEnd();

astr3--;

if (astr3<-500) {

astr3 = 0;

lrIndex3 = lrIndex;

}

//KIll check asteroid3

if ((abs(lrIndex - lrIndex3)<8) && (astr3 + 500<10)) {

start = 0;

gv = 1;

}

printf("gv:%d\t", gv);

//Print Score inside score board

char buffer[50];

sprintf\_s(buffer, "SCORE: %d", score);

glColor3f(0.000, 1.000, 0.000);

renderBitmapString(12, 78, (void \*)font3, buffer);

//Speed Print inside score board

char buffer1[50];

sprintf\_s(buffer1, "SPEED:%dKm/h", FPS);

glColor3f(0.000, 1.000, 0.000);

renderBitmapString(12, 90 - 22, (void \*)font3, buffer1);

//level Print inside score board

if (score % 50 == 0) {

int last = score / 50;

if (last != level) {

level = score / 50;

FPS = FPS + 2;

}

}

char level\_buffer[50];

sprintf\_s(level\_buffer, "LEVEL: %d", level);

glColor3f(0.000, 1.000, 0.000);

renderBitmapString(12, 90 - 32, (void \*)font3, level\_buffer);

if (gv == 1)

{

glClearColor(0.009, 0.0077, 0.397845, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.000, 1.000, 1.070);

renderBitmapString(190, 260 + 10, (void \*)font1, "GAME OVER");

renderBitmapString(139, 220 + 10, (void \*)font1, "SPACEBAR TO RESTART");

renderBitmapString(179, 190 + 10, (void \*)font1, "ESC TO EXIT");

}

for (j = 0; j <= 1000000; j++)

;

glutSwapBuffers();

glutPostRedisplay();

glFlush();

}}

//Special functions

void spe\_key(int key, int x, int y)

{

switch (key) {

case GLUT\_KEY\_DOWN://FPS is initially 50

if (FPS>(50 + (level \* 2)))

FPS = FPS - 2;

break;

case GLUT\_KEY\_UP:

FPS = FPS + 2;

break;

case GLUT\_KEY\_LEFT:

if (lrIndex >= -100) {

lrIndex = lrIndex - (FPS / 10);//(FPS/10)gives speed of left movement

if (lrIndex<-100) {

lrIndex = -101;

}

}

break;

case GLUT\_KEY\_RIGHT://44 indictaes border of the road

if (lrIndex <= 150) {

lrIndex = lrIndex + (FPS / 10);

if (lrIndex>600) {

lrIndex = 45;

}

}

break;

default:

break;

}

}

void processKeys(unsigned char key, int x, int y) {

switch (key)

{

case ' ':

if (start == 0) {

start = 1;

gv = 0;

FPS = 50;

lrIndex = 0;

astr1 = 0;

lrIndex1 = 0;

astr2 = +35;

lrIndex2 = 0;

astr3 = +70;

lrIndex3 = 0;

score = 0;

level = 0;

}

break;

case 27:

exit(0);

break;

default:

break;

}

}

void moon(float radius)//moon

{

glBegin(GL\_POLYGON);

for (int i = 0; i <= 359; i++)

{

float degInRad = i\*DEG2RAD;

glVertex2f(300 + f + cos(degInRad)\*radius, 500 - t + (sin(degInRad))\*radius);

//100,100 specifies centre of the circle

}

glEnd();

t = t + 0.1;

f = f + 0.1;

}

void myinit()

{

glClearColor(0.196078, 0.6, 0.8, 1.0);

glPointSize(1.0);

gluOrtho2D(0.0, 499.0, 0.0, 499.0);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("rocket");

glutIdleFunc(display1);

glutDisplayFunc(display1);

glutSpecialFunc(spe\_key);

glutKeyboardFunc(processKeys);

myinit();

glutMainLoop();

return 0;

}

**CHAPTER 5**

**RESULT AND ANALYSIS**

**5.1 User interface**

**glutKeyboardFunc()**

Input and interaction form the most important part of computer graphics. The mouse and the keyboard are used to interact with the application and in this program we are using keyboard as the user interface. The function used is :

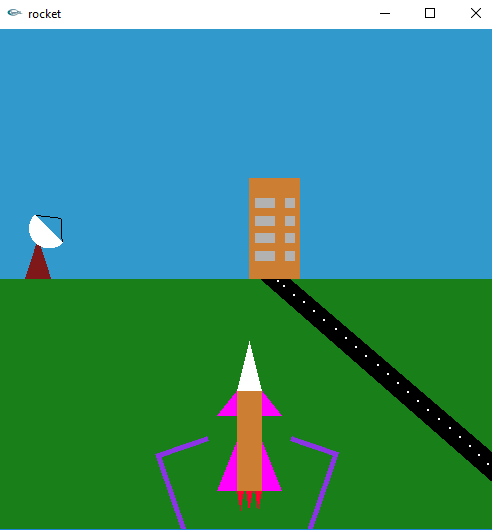
**void processKeys(unsigned char key, int, int)**

Each and every movement of the output view of this particular project has been completely controlled by the keyboard function. According to the keyboard inputs the corresponding actions are executed.

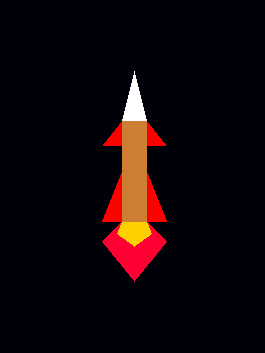
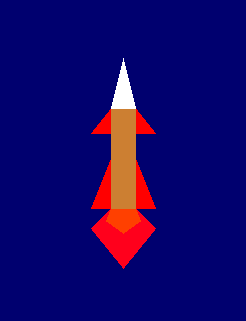
Following are the Keyboard movements used in the Rocket Mission

* Left arrow key, the rocket moves to the left.
* Right arrow key, the rocket moves to the right.
* Top arrow key, the rocket speed increases.
* Bottom arrow key, the rocket speed decreases.
* Esc button to exit the game.
* Spacebar button to restart the game.

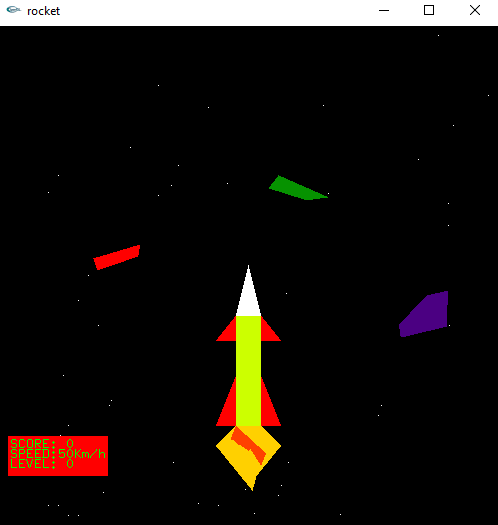
**5.2 Snapshots**

****

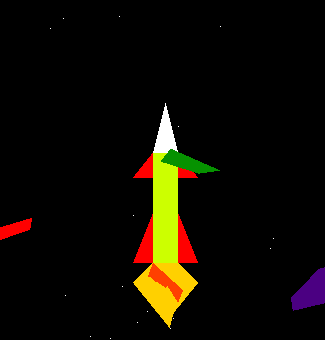
**Figure 5.1: snapshot of the rocket launch**

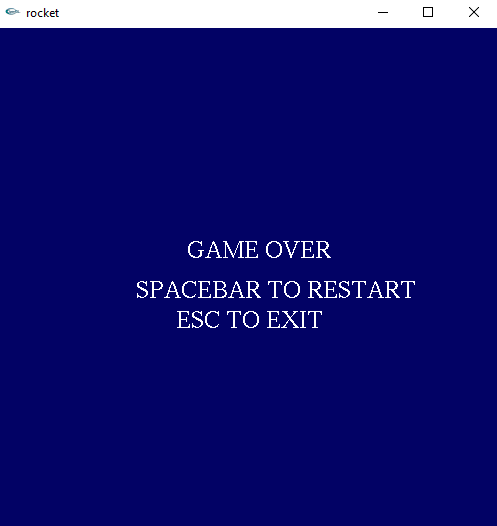
 

**Figure 5.2: snapshot showing the rocket moving into space from earths atmosphere**



**Figure 5.3: snapshot of rocket in outer-space and asteroids in its path**

  
 **Figure 5.4: Snapshot of rocket collision with asteroid**



**Figure 5.5: Snapshot of screen after collision**

**CHAPTER 6**

**CONCLUSION**

The design and development of the project on a windows platform gave me an insight

at the problems that I would be faced by when taking a look at real world software engineering.

The experience I gained by designing this project is valuable. This has helped me in exploring

the different galleries of OpenGL function which I was not aware of, I come to know the

numerous inbuilt functions that were available in OpenGL.

This application gave me an opportunity to explore the graphic abilities of OpenGL

and C++. We can further improve this project depending on the needs and views of the users.

We can include many more concepts into this project to make it even more user-friendly.

This project has enhanced my programming and imaginative abilities to a great extent.

**6.1 Future enhancement**

Any system cannot achieve one hundred percent perfection at the first time and this is

no different from the universal truth. The different menus can be added to display at the runtime, also shortcut keys can be assigned to ease the access of various menu items. The following may be considered as future enhancements which might make the system more user friendly, flexible and more interactive.

By implementing a project using OpenGL I came to know how to use the functions such as rotation, translation and scaling. These functions helped implementing a living affect to the object by providing movements and I have also learnt how to view an object in different views.

**BIBLIOGRAPHY**

**BOOKS**

1. Edward Angel: Interactive Computer Graphics A Top-Down Approach with OpenGL, 5th Edition, Person Education, 2008.
2. Donald Hearn and Pauline Baker: Computer Graphics-OpenGL Version, 3rd Edition, Pearson Education, 2004.

**Websites**

* [www.openGLprojects.com](http://www.openGLprojects.com)
* [www.opengl.org](http://www.opengl.org)
* [www.google.com](http://www.google.com)