VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi-590018, Karnataka, INDIA

PROJECT REPORT

On

"SIMULATION OF PSLV C-37 BY ISRO"

Submitted in partial fulfillment of the requirements for the VI Semester

Computer Graphics and Visualization Lab (10CSL67)

Bachelor of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

For the Academic year

2017-2018

BY

SUSHMA	1PE14CS156
SHASHANK SANKET	1PE14CS181

Department of Computer Science and Engineering

PESIT BANGALORE SOUTH CAMPUS

HOSUR ROAD

BENGALURU-560100

PESIT BANGALORE SOUTH CAMPUS

HOSUR ROAD

BENGALURU-560100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitle ***"SIMULATION OF PSLV C-37 BY ISRO"*** *is a bonafide work carried out by* ***SUSHMA*** *and* ***SHASHANK SANKET*** *bearing USN* ***1PE14CS156 and 1PE14CS181*** respectively, *in Computer Graphics and Visualization Lab(10CSL67) for the 6th Semester in partial fulfillment for the award of Degree of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University ,Belagavi during the year 2017-2018.*

---------------------------	---------------------------
Signature of the guide	***Signature of the HOD***
Ms. Neeta	**Prof. Sandesh B J**
Assistant Professor CSE	HOD and Associate Professor Dept. Dept of Dept. of CSE
PESIT BSC, Bengaluru.	PESIT BSC, Bengaluru.

ACKNOWLEDGEMENTS

Firstly, we would like to thank our project coordinator **Ms. Neeta, Assistant Professor** for guiding us through the various stages of this project. Without their assistance, this project would not have been possible.

We are immensely grateful to the **Professor/Head of the Department** of Computer Science and Engineering, **Dr. Sandesh B J**, for ensuring that we completed the project without any hassles.

We would also like to thank our **Principal/Director Dr. J. Surya Prasad**, who gave us this wonderful opportunity and platform to experiment with our existing knowledge of the subject.

We would like to thank our college for providing this opportunity to create this interesting game and providing us with the necessary infrastructure to successfully to create this project. Support of friends and family was also of utmost importance for the completion of this project.

SUSHMA (1PE14CS156)

SHASHANK SANKET (1PE14CS181)

ABSTRACT

The rocket launching project is a 2D animation of a satellite launching process.

In our project rocket will be static prior to launching. When a launch is performed through interactions, the countdown begins for the launch. The countdown is 8 to 0, once the countdown is finished rocket is launched. In ground stage1 all the surrounding is displayed as per the natural scene near ISRO launching pad. Once the rocket is launched, actual animation starts here. Rocket starts moving into the sky, after sometime it reaches the space. At this time background color is changed to black with twinkling stars around the space. The next phase of the rocket launch is dismantling stage2, in this stage the first fuel supplier will be dismantled. Now rocket starts moving with the help of second fuel supplier. The next stage will be dismantling stage2, here the fuel supplier 2 will be dismantled. In the last and stage3 the rocket pass by moon and at last it ends it journey in planet Mars which is last stage of simulation.

Each stage is demonstrated with real situation of rocket launching vehicle used by ISRO and to distinguish between the stages real situation background color are used to make it a real simulation.

TABLE OF CONTENTS

1. INTRODUCTION TO OPENGL

OPENGL:

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes. Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors etc.) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

At its most basic level OpenGL is a specification, meaning it is simply a document that describes a set of functions and the precise behaviors that they must perform. From this specification, hardware vendors create implementations. Libraries of functions created to match the functions stated in the OpenGL specification, making use of hardware acceleration where possible. Hardware vendors have to meet specific tests to be able to qualify their implementation as an OpenGL implementation.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

OpenGL serves two main purposes:

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

OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives. Prior to the introduction of OpenGL 2.0, each stage of the pipeline performed a fixed function and was configurable only within tight limits. OpenGL 2.0 offers several stages that are fully programmable using GLSL.

GLUT:

GLUT is the OpenGL Utility Toolkit. GLUT is used in the development of OpenGL applications. OpenGL is a graphics library which means that it can do all sorts of things with geometric forms, lightings, transformations etc. But OpenGL was designed to be portable so it can be used within various environments / operating systems without changing the OpenGL code.

GLUT provides a utility that creates a window, in which the output of an OpenGL program can be displayed.

A very important aspect of GLUT is that it uses call hack functions as event handlers. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so one can write a single OpenGL program that works across all PC and workstation OS platforms.

GLUT is designed for constructing small to medium sized OpenGL programs. GLUT is well suited to learning OpenGL and developing simple OpenGL applications.

2. SOFTWARE AND HARDWARE SPECIFICATION

Software:

- Linux 2.6.27 or later
- OpenGL and GLUT libraries
- CodeBlocks IDE

Hardware:

- Pentium 4 or Equivalent Processor.
- 256 MB RAM
- 5 MB Disk Space
- Intel GMA 850 or better display adaptor.
- Mouse and Keyboard input devices.

Platform and Tools:

- The project has been carried out in Linux platform.
- Built using CodeBlocks IDE with GLUT libraries.
- Keyboard interface is required for this project.

PROJECT DESCRIPTION

PSLV Simulations by ISRO are designed to accurately model the flow of real world Satellite launching vehicles used across world. They form a complex system that is open. Shows emergent phenomena, non-linear relationships and can contain feedback loops.

The Rocket Launching takes place from the ground .The ground just vanishes and sky color changes instantly. This starts from the base of launch on the earth having buildings and communication tower shown in background then sky color is blue(sky blue) then in space background have dark color with stars.

After the launch ground moves down and left and right supporting structures rotates the sky turns black from blue stars appears and the moon.

In this simple graphics in C we use the logic of launching a space shuttle its journey into a space crossing the mars. Different stages of rocket propulsion is avoided due to its complication. This computer graphics program in C simply demonstrate the space shuttle launch, its way to leave the earth entering space and journey into the space.

4. API DESCRIPTION

1) glutInit(&argc, argv)

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

argc: A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argc will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv : The program's unmodified argv variable from main. Like argc, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

2) glutInitDisplayMode(unsigned int mode)

The initial display mode is used when creating top-level windows, sub windows and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

Parameters :

GLUT_RGB : Bit mask to select an RGBA mode window. This is the default if neither GLUT _RGBA nor GLUT _INDEX are specified.

GLUT_SINGLE : Bit mask to select a single buffered window. This is the default if neither GLUT_ DOUBLE nor GLUT_SINGLE are specified.

3) glutInitWindowSize(int width, int height) :

Windows created by glutCreateWindow will be requested to be created with the current size.

width: Width in pixels.

height : Height in pixels.

4) glutCreateWindow(const char *title):

It creates and opens OpenGL window with the title passed as the argument.

5) glutInitWindowPosition(int x, int y) :

Windows created by glutCreateWindow will be requested to be created with the current initial position.

x: Window X location in pixels.

y : Window Y location in pixels.

The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine. The initial value of the initial window size GLUT state is 300 by 300.

6) glutCreateWindow(char *name) :

glutCreateWindow creates a top-level window. The *name* will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

7) glutDisplayFunc(void(*func)(void)):

glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called.

Before the callback, the current window is set to the window needing to be redisplayed. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback.

8) glLoadIdentity (void):

Used to initialize the current transform matrix into the identity transform.

9) glutKeyboardFunc(void (*func)(unsigned char key, int x, int y):

glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

Func: The new keyboard callback function

10) glutMainLoop( ) :

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call any callbacks that have been registered.

11) glClearColor(GLfloat red,GLfloat green,GLfloat blue) :

glClearColor specifics the red, green, blue and alpha values used by glClear to clear the color buffers. Values specified by gIClearColor are clamped to the range [0,1].

red , green , blue , alpha : Specify the red, green, blue and alpha values used when the color buffers are cleared. The initial values are all 0.

12) glutPostRedisplay(void):

Mark the normal plane of current window as needing to be redisplayed.

13) gluOrtho2D(GLdouble left,GLdouble right, GLdouble bottom, GLdouble top) :

gluOrtho2D sets up a two-dimensional orthographic viewing region .

left, right :

Specify the coordinates for the left and right vertical clipping planes.

bottom, top :

Specify the coordinates for the bottom and top for horizontal clipping planes.

14) glClear(GLbitfieldmask) :

glClear takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

Mask :

Bitwise OR of masks that indicate the buffer to be cleared. The four masks are GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_BUFFER_BIT, and GL_STENCIL_BUFFER_BIT.

15) glColor3f(GLfloat red,GLfloat green,GLfloat blue):

glColor sets a new three valued RGB color. Current color values are stored in floating-point format. Unsigned integer color components are linearly mapped to floating-point values such that the largest representable value maps to 1.0 (full intensity), and 0 maps to 0.0 (zero intensity).

red: The new red value for the current color.

green: The new green value for the current color.

blue: The new blue value for the current color.

16) glBegin(GLenum mode) :

glBegin and glEnd delimit the vertices that define a primitive or a group of primitives. glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted. Regardless of the value chosen for mode, there is no limit to the number of vertices that can he defined between glBegin and glEnd. Lines, triangles, quadrilaterals, and polygons that are incompletely specified are not drawn. Incomplete specification results when either too few vertices are provided to specify even a single primitive or when an incorrect multiple of vertices is specified. The incomplete primitive is ignored, the rest are drawn.

Mode :

Specifies the primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. Ten symbolic constants are accepted:

GL_POINTS , GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP , GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN , GL_QUADS, GL_QUAD_STRIP and GL_POLYGON.

17) glVertex2i(GLint x , GLint y) :

glVertex commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, and texture coordinates are associated with the vertex when glVertex is called.

x, y :

Specifies x and y coordinates

18) glMatrixMode (GLenum mode):

It switches matrix mode between the two matrices-

- MODEL_VIEW (GL_MODELVIEW)
- PROJECTION (GL_PROJ ECTION)

19) glutSwapBuffers( ):

We can swap the front and back buffers that will form the application programs.

20) glViewport (GLint x, GLint y, GLsizei width, GLsizei height):

It specifies that the viewport will have lower left corner (x,y) in screen coordinates and will be width pixels wide and height pixels high.

21) glFlush( ):

Different GL implementations buffer commands in several different locations, including network buffers, and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine.

PROJECT IMPLEMENTATION

This project was implemented completely on OpenGL. We have used a lot of user defined functions which are listed below:

1. Main function

This main function is capable of handling the arguments given in the argument list at the command prompt as we have used variable ‘argc’ for total number of arguments and ‘argv’ for the array of argument list.

Main function initializes the Display Mode, Window Size and position. Then it invokes the display function within glutDisplayFunc() as the call back function.

2. Displaying text on the screen

The setFont() function is used to set the type of the font we are using. The drawstring() function takes the position of the string to be displayed on the screen in X Y Z coordinates and the string to display.

3. Void Init( )

Initializes the Window and creates the view volume, creates orthographic parallel projection and model view.

4. Void semicircle( )

Used to draw satellite tower in space station.

5. Void moon( )

Used to draw moon in space background.

6. Void staticrocket(void)

Used to draw the rocket initially on the ground before ignition at space station center.

7. Void motionrocket( )

Used to draw the rocket that is ignited and in motion toward space

8. Void stars( )

Used to show the stars in space and also shows the twinkling effect.

9. Void mars( )

Used to draw planet mars in red color.

10. Displaying text on the screen

The setFont() function is used to set the type of the font we are using. The drawstring() function takes the position of the string to be displayed on the screen in X Y Z coordinates and the string to display.

11. Update functions

Update function is used for the transformation of the objects this function is invoked in display function inside glutTimerFunc(50,update,0). This function calls the update every 50 milliseconds hence the variable 'a', 'b' are all modified then when we translate the object with this points we will see the animation effect.

SOURCE CODE

main.c

```
#include<GL/glut.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<string.h>
const float DEG2RAD = 3.14159/180;
float i,j,count=0,count1=0,count3=0,flag=0,flag1=0,t=0,f=0,flag3=0;
void moon(float radius);

void semicircle(float radius,float u,float v)
{

        glColor3f(0.78,0.78 ,0.78);
  glBegin(GL_POLYGON);

  for (int i=135; i<=315; i++)
  {
    float degInRad = i*DEG2RAD;
    glVertex2f(u+cos(degInRad)*radius,v+(sin(degInRad))*radius);//100,100 specifies centre of
the circle
  }

  glEnd();
}
void moon(float radius)
{
   glColor3f(1.0,1.0,1.0);
   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+0.2;
  f=f+0.2;
```

```
}

void mars(float radius)
{
    glColor3f(1.0,0.3,0.3f);
    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+0.2;
   f=f+0.2;
}

void text()
{
char menu[90],a[50],b[50],c[50],d[50],e[50];
int len0,len1,len2,len3,len4,len5;
strcpy(menu,"     CG PROJECT");
strcpy(a,"          PSLV -C37 ISRO");
strcpy(b,"Shashank sanket");
strcpy(c,"1PE14CS181");
strcpy(d,"Sushma");
strcpy(e,"1PE14CS156");
len0=strlen(menu);
len1=strlen(a);
len2=strlen(b);
len3=strlen(c);
len4=strlen(d);
len5=strlen(e);
glColor3f(1.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluOrtho2D(0,600,0,600);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
glRasterPos2i(450,550);
for(int i=0;i<len0;++i)
{
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,menu[i]);
}
glRasterPos2i(400,450);
for(int i=0;i<len1;++i)
{
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,a[i]);
}
  glColor3f(0.0,1.0,0.0);
  glRasterPos2i(20,580);
for(int i=0;i<len2;++i)
{
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,b[i]);
  }
  glRasterPos2i(30,560);
for(int i=0;i<len3;++i)
{
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,c[i]);
}
glRasterPos2i(20,540);
for(int i=0;i<len4;++i)
{
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,d[i]);
}
glRasterPos2i(30,520);
for(int i=0;i<len5;++i)
{
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,e[i]);
}
glPopMatrix();
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
}
void staticrocket(void)
{
            glColor3f(0.8,0.298039 ,0.26078);
            glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

            glColor3f(0.7,0.7,0.7);
        glBegin(GL_POLYGON);//rocket space ground
                    glVertex2f(290,230);
                    glVertex2f(700, 230);
                    glVertex2f(700, 700);
                    glVertex2f(200, 700);
                    glEnd();
             glColor3f(0.0,0.5,0.5);//rocket top
            glBegin(GL_POLYGON);
            glVertex2f(350,400);
            glVertex2f(365,440);
            glVertex2f(380,400);
            glEnd();

    glColor3f(1.2,1.2,1.2);//rocket base
            glBegin(GL_POLYGON);
            glVertex2f(350,300);
            glVertex2f(380,300);
            glVertex2f(380,400);
            glVertex2f(350, 400);
            glEnd();
       glColor3f(1.0,0.0,0.0); //band color
                glBegin(GL_POLYGON);
                glVertex2f(350,390);
                glVertex2f(380,390);
                glVertex2f(350,380);
                glVertex2f(380,380);
                glEnd();
       glColor3f(0.8,0.4,0.0);// band color
                glBegin(GL_POLYGON);
                glVertex2f(350,370);
                glVertex2f(380,370);
                glVertex2f(350,360);
                glVertex2f(380,360);
                glEnd();
       glColor3f(1.0,0.0,0.0);//band color
                glBegin(GL_POLYGON);
                glVertex2f(350,350);
                glVertex2f(380,350);
                glVertex2f(350,340);
                glVertex2f(380,340);
                glEnd();


    glColor3f(1.0,0.0,0.0);
    glBegin(GL_POLYGON);//left_side_top
    glVertex2f(350,400);
    glVertex2f(330,360);
    glVertex2f(350,360);
    glEnd();
         glBegin(GL_POLYGON);//left_side_bottom
    glVertex2f(350,330);
    glVertex2f(330,300);
```

```
glVertex2f(350,300);
glEnd();
        glBegin(GL_POLYGON);//right_side_bottom
glVertex2f(380,330);
glVertex2f(400,300);
glVertex2f(380,300);
glEnd();
        glBegin(GL_POLYGON);//right_side_top
glVertex2f(380,400);
glVertex2f(400,360);
glVertex2f(380,360);
glEnd();
                glBegin(GL_POLYGON);//bottom_1_exhaust
glVertex2f(350,300);
glVertex2f(330,290);
glVertex2f(365,275);
    glVertex2f(400,290);
    glVertex2f(380,300);
glEnd();
    glColor3f(1.0,1.0,0.0);
        glBegin(GL_POLYGON);//bottom_2_exhaust
glVertex2f(355,300);
glVertex2f(350,290);
glVertex2f(365,280);
glVertex2f(380,290);
glVertex2f(375,300);
glEnd();
    glColor3f(0.1,0.0,0.0);
glBegin(GL_POLYGON);//left_stand_holder
glVertex2f(350,340);
glVertex2f(300,340);
glVertex2f(300,260);
glVertex2f(305,260);
glVertex2f(305,335);
    glVertex2f(350,335);
glEnd();
glBegin(GL_POLYGON);
                glVertex2f(380,340);//right_stand_holder
    glVertex2f(430,340);
    glVertex2f(430,260);
    glVertex2f(425,260);
    glVertex2f(425,335);
glVertex2f(380,335);
glEnd();
```

```
glColor3f(0.8,0.298039 ,0.26078);
glBegin(GL_POLYGON);//house 1
glVertex2f(0.0,0.0);
glVertex2f(90.0,0.0);
glVertex2f(90.0,210.0);
glVertex2f(0.0,210.0);
glEnd();

glColor3f(0.7,0.7,0.7);
glBegin(GL_POLYGON);//HOUSE 1A
        glVertex2f(10.0,20.0);
        glVertex2f(10.0,50.0);
        glVertex2f(40.0,50.0);
        glVertex2f(40.0,20.0);
        glEnd();
glBegin(GL_POLYGON);//HOUSE 1B
        glVertex2f(10.0,70.0);
        glVertex2f(40.0,70.0);
        glVertex2f(40.0,100.0);
        glVertex2f(10.0,100);
        glEnd();

glBegin(GL_POLYGON);//HOUSE 1C
        glVertex2f(10.0,120.0);
        glVertex2f(40.0,120.0);
        glVertex2f(40.0,150.0);
        glVertex2f(10.0,150);
        glEnd();

glBegin(GL_POLYGON);//HOUSE 1D
        glVertex2f(10.0,170.0);
        glVertex2f(40.0,170.0);
        glVertex2f(40.0,200.0);
        glVertex2f(10.0,200);
        glEnd();

glColor3f(0.8,0.298039 ,0.26078);

glBegin(GL_POLYGON);//house2
glVertex2f(310,0);
glVertex2f(500,0);
glVertex2f(500,210);
glVertex2f(310,210);
glEnd();
```

```
int x=25;
int y=7;

glColor3f(0.8,0.198039 ,0.26078);
                glBegin(GL_POLYGON);//house 1
                glVertex2f(0.0+x,0.0+y);
                glVertex2f(90.0+x,0.0+y);
                glVertex2f(90.0+x,210.0+y);
                glVertex2f(0.0+x,210.0+y);
                glEnd();

glColor3f(0.7,0.7,0.7);
                glBegin(GL_POLYGON);//HOUSE 1A
                          glVertex2f(10.0+x,20.0+x);
                          glVertex2f(10.0+x,50.0+x);
                          glVertex2f(40.0+x,50.0+x);
                          glVertex2f(40.0+x,20.0+x);
                          glEnd();
                glBegin(GL_POLYGON);//HOUSE 1B
                          glVertex2f(10.0+x,70.0+x);
                          glVertex2f(40.0+x,70.0+x);
                          glVertex2f(40.0+x,100.0+x);
                          glVertex2f(10.0+x,100.0+x);
                          glEnd();

                glBegin(GL_POLYGON);//HOUSE 1C
                          glVertex2f(10.0+x,120.0+x);
                          glVertex2f(40.0+x,120.0+x);
                          glVertex2f(40.0+x,150.0+x);
                          glVertex2f(10.0+x,150+x);
                          glEnd();

                /*glBegin(GL_POLYGON);//HOUSE 1D
                          glVertex2f(10.0+x,170.0+x);
                          glVertex2f(40.0+x,170.0+x);
                          glVertex2f(40.0+x,200.0+x);
                          glVertex2f(10.0+x,200+x);
                          glEnd();*/

                glColor3f(0.6,0.228039 ,0.21078);

                glBegin(GL_POLYGON);//house2
                glVertex2f(310+x,0+x);
                glVertex2f(500+x,0+x);
```

```
                    glVertex2f(500+x,210+x);
                    glVertex2f(310+x,210+x);
                    glEnd();



int x1=60;
int y1=14;

   glColor3f(0.7,0.298039 ,0.24078);
                    glBegin(GL_POLYGON);//house 1
                    glVertex2f(0.0+x1,0.0+y1);
                    glVertex2f(90.0+x1,0.0+y1);
                    glVertex2f(90.0+x1,210.0+y1);
                    glVertex2f(0.0+x1,210.0+y1);
                    glEnd();

   glColor3f(0.7,0.7,0.7);
                    glBegin(GL_POLYGON);//HOUSE 1A
                              glVertex2f(10.0+x1,20.0+x1);
                              glVertex2f(10.0+x1,50.0+x1);
                              glVertex2f(40.0+x1,50.0+x1);
                              glVertex2f(40.0+x1,20.0+x1);
                              glEnd();
                    glBegin(GL_POLYGON);//HOUSE 1B
                              glVertex2f(10.0+x1,70.0+x1);
                              glVertex2f(40.0+x1,70.0+x1);
                              glVertex2f(40.0+x1,100.0+x1);
                              glVertex2f(10.0+x1,100.0+x1);
                              glEnd();

                    glBegin(GL_POLYGON);//HOUSE 1C
                              glVertex2f(10.0+x1,120.0+x1);
                              glVertex2f(40.0+x1,120.0+x1);
                              glVertex2f(40.0+x1,150.0+x1);
                              glVertex2f(10.0+x1,150+x1);
                              glEnd();

             glBegin(GL_POLYGON);//HOUSE2 1B
                    glVertex2f(340.0,70.0);
                    glVertex2f(380.0,70.0);
                    glVertex2f(380.0,100.0);
                    glVertex2f(340.0,100);
                    glEnd();

             glBegin(GL_POLYGON);//HOUSE2 1C
```

```
        glVertex2f(340.0,120.0);
        glVertex2f(380.0,120.0);
        glVertex2f(380.0,150.0);
        glVertex2f(340.0,150);
        glEnd();

glBegin(GL_POLYGON);//HOUSE2 1D
        glVertex2f(340.0,170.0);
        glVertex2f(380.0,170.0);
        glVertex2f(380.0,200.0);
        glVertex2f(340.0,200);
        glEnd();
        glColor3f(0.7,0.7,0.7);

glBegin(GL_POLYGON);//HOUSE2 2B
        glVertex2f(420.0,70.0);
        glVertex2f(460.0,70.0);
        glVertex2f(460.0,100.0);
        glVertex2f(420.0,100);
        glEnd();

glBegin(GL_POLYGON);//HOUSE2 2C
        glVertex2f(420.0,120.0);
        glVertex2f(460.0,120.0);
        glVertex2f(460.0,150.0);
        glVertex2f(420.0,150);
        glEnd();

glBegin(GL_POLYGON);//HOUSE2 2D
        glVertex2f(420.0,170.0);
        glVertex2f(460.0,170.0);
        glVertex2f(460.0,200.0);
        glVertex2f(420.0,200);
        glEnd();


        glColor3f(0.4,0.168039 ,0.04078); //house 2
        glBegin(GL_POLYGON);
        glVertex2f(110.0,0.0);
        glVertex2f(190.0,0.0);
        glVertex2f(190.0,150.0);
        glVertex2f(110.0,150.0);
        glEnd();
        glColor3f(0.7,0.7,0.7);
        glBegin(GL_POLYGON);//HOUSE 2A
```

```
                glVertex2f(130.0,5.0);
                glVertex2f(144.0,5.0);
                glVertex2f(144.0,40.0);
                glVertex2f(130.0,40.0);
                glEnd();

        glColor3f(0.7,0.7,0.7);
        glBegin(GL_POLYGON);//HOUSE 2B
                glVertex2f(130.0,55.0);
                glVertex2f(144.0,55.0);
                glVertex2f(144.0,95.0);
                glVertex2f(130.0,95.0);
                glEnd();

        glColor3f(0.7,0.7,0.7);
        glBegin(GL_POLYGON);//HOUSE 2C
                glVertex2f(130.0,110.0);
                glVertex2f(144.0,110.0);
                glVertex2f(144.0,145.0);
                glVertex2f(130.0,145.0);
                glEnd();


int x2=12;
int y2=8;
glColor3f(1.0,1.0 ,0.5); //house 2
        glBegin(GL_POLYGON);
        glVertex2f(110.0+x2,0.0+y2);
        glVertex2f(190.0+x2,0.0+y2);
        glVertex2f(190.0+x2,150.0+y2);
        glVertex2f(110.0+x2,150.0+y2);
        glEnd();
        glColor3f(0.7,0.7,0.7);
        glBegin(GL_POLYGON);//HOUSE 2A
                glVertex2f(130.0+x2,5.0+x2);
                glVertex2f(144.0+x2,5.0+x2);
                glVertex2f(144.0+x2,40.0+x2);
                glVertex2f(130.0+x2,40.0+x2);
                glEnd();

        glColor3f(0.7,0.7,0.7);
        glBegin(GL_POLYGON);//HOUSE 2B
                glVertex2f(130.0+x2,55.0+x2);
                glVertex2f(144.0+x2,55.0+x2);
                glVertex2f(144.0+x2,95.0+x2);
```

```
                        glVertex2f(130.0+x2,95.0+x2);
                        glEnd();

                glColor3f(0.7,0.7,0.7);
                glBegin(GL_POLYGON);//HOUSE 2C
                        glVertex2f(130.0+x2,110.0+x2);
                        glVertex2f(144.0+x2,110.0+x2);
                        glVertex2f(144.0+x2,145.0+x2);
                        glVertex2f(130.0+x2,145.0+x2);
                        glEnd();

        //banner of PSLV.................................
        glColor3f(1.0,1.0,0.0);
                glBegin(GL_POLYGON);
      glVertex2f(410,430);
      glVertex2f(560,430);
      glVertex2f(560,470);
      glVertex2f(410,470);
      glEnd();

      glBegin(GL_LINES);
      glVertex2f(411,430);
      glVertex2f(411,360);
      glEnd();

      glBegin(GL_LINES);
      glVertex2f(560,430);
      glVertex2f(560,360);
      glEnd();

semicircle(60.0, 110 ,450);
                glColor3f(0.7,0.168039 ,0.24078); // tower
          glBegin(GL_LINE_LOOP);
          glVertex2f(20.0,240.0);
          glVertex2f(120.0,240.0);
          glVertex2f(80.0,400.0);
          glVertex2f(60.0,400.0);
          glEnd();
          glColor3f(0.7,0.168039 ,0.24078); // tower
          glBegin(GL_LINE_LOOP);
          glVertex2f(17.5,240);
          glVertex2f(122.5,240);
          glVertex2f(82.5,400.0);
          glVertex2f(57.5,400.0);
          glEnd();
```

```
    glColor3f(0.7,0.168039 ,0.24078);
            glBegin(GL_LINE_STRIP);
            glVertex2f(20.0,240);
            glVertex2f(112.5,270.0);
            glVertex2f(27.5,270.0);
            glVertex2f(105,300.0);
            glVertex2f(35,300.0);
                    glVertex2f(97.5,330.0);
                    glVertex2f(42.5,330.0);
                    glVertex2f(90.5,360.0);
                    glVertex2f(50.5,360.0);
                    glVertex2f(80.5,400.0);
            glEnd();


             glColor3f(0.7,0.168039 ,0.24078);
       glBegin(GL_LINE_STRIP);
       glVertex2f(20.0,240);
            glVertex2f(112.5,272.5);
            glVertex2f(27.5,270);
            glVertex2f(105,302.5);
            glVertex2f(35,300.0);
            glVertex2f(97.5,332.5);
            glVertex2f(42.5,330.0);
            glVertex2f(90.5,362.5);
            glVertex2f(50.5,360.0);
            glVertex2f(80.5,400.0);
            glEnd();

            glColor3f(0.1,0.0,0.1);

            glBegin(GL_QUADS);//Road
            glVertex2f(220,0);
            glVertex2f(300,0);
            glVertex2f(170,700);
            glVertex2f(120,700);
            glEnd();

            glColor3f(1.0,1.0,1.0);//road strips
            glBegin(GL_LINES);
            glVertex2f(0,0);
            glVertex2f(245,0);
            glVertex2f(240,40);
            glVertex2f(235,80);
            glVertex2f(230,120);
```

```
                    glVertex2f(225,160);
                    glVertex2f(220,200);
                    glVertex2f(215,240);
                    glVertex2f(210,280);
                    glVertex2f(205,320);
                    glVertex2f(200,360);
                    glVertex2f(195,400);
                    glVertex2f(190,440);
                    glVertex2f(185,480);
                    glVertex2f(180,520);
                    glVertex2f(175,560);
                    glVertex2f(170,600);
                    glVertex2f(165,640);
                    glVertex2f(160,680);
                    glVertex2f(155,720);
                    glVertex2f(140,760);
                    glVertex2f(145,800);
                    glEnd();
                    text();
for(j=0;j<=1000000;j++)
                    ;
          glutSwapBuffers();
          glutPostRedisplay();

                    glFlush();
}

void motionrocket()
{
          count++;


for(i=90;i<=140;i++)
{
    if(count>=8)
           {
                              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(0.196078  ,0.6 ,0.8,1.0);
          glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
           }
           if(count>=8)
           moon(40.0);
           if(count>=16)
           mars(180.0);
           if(count>=10)



glColor3f(0.0,0.5,0.5);//rocket top
           glBegin(GL_POLYGON);
           glVertex2f(350,400+i);
           glVertex2f(365,440+i);
           glVertex2f(380,400+i);
           glEnd();

     glColor3f(1.2,1.2,1.2);//rocket base
           glBegin(GL_POLYGON);
           glVertex2f(350,300+i);
           glVertex2f(380,300+i);
           glVertex2f(380,400+i);
           glVertex2f(350, 400+i);
           glEnd();
          glColor3f(1.0,0.0,0.0); //band color
                    glBegin(GL_POLYGON);
                    glVertex2f(350,390+i);
                    glVertex2f(380,390+i);
                    glVertex2f(350,380+i);
                    glVertex2f(380,380+i);
                    glEnd();
          glColor3f(0.8,0.4,0.0);// band color
                    glBegin(GL_POLYGON);
                    glVertex2f(350,370+i);
                    glVertex2f(380,370+i);
                    glVertex2f(350,360+i);
                    glVertex2f(380,360+i);
```

```
            glEnd();
    glColor3f(1.0,0.0,0.0);//band color
            glBegin(GL_POLYGON);
            glVertex2f(350,350+i);
            glVertex2f(380,350+i);
            glVertex2f(350,340+i);
            glVertex2f(380,340+i);
            glEnd();


glColor3f(1.0,0.0,0.0);
glBegin(GL_POLYGON);//left_side_top
glVertex2f(350,400+i);
glVertex2f(330,360+i);
glVertex2f(350,360+i);
glEnd();
      glBegin(GL_POLYGON);//left_side_bottom
glVertex2f(350,330+i);
glVertex2f(330,300+i);
glVertex2f(350,300+i);
glEnd();
      glBegin(GL_POLYGON);//right_side_bottom
glVertex2f(380,330+i);
glVertex2f(400,300+i);
glVertex2f(380,300+i);
glEnd();
      glBegin(GL_POLYGON);//right_side_top
glVertex2f(380,400+i);
glVertex2f(400,360+i);
glVertex2f(380,360+i);
glEnd();
            glBegin(GL_POLYGON);//bottom_1_exhaust
glVertex2f(350,300+i);
glVertex2f(330,290+i);
glVertex2f(365,275+i);
   glVertex2f(400,290+i);
   glVertex2f(380,300+i);
glEnd();
   glColor3f(1.0,1.0,0.0);
      glBegin(GL_POLYGON);//bottom_2_exhaust
glVertex2f(355,300+i);
glVertex2f(350,290+i);
glVertex2f(365,280+i);
glVertex2f(380,290+i);
glVertex2f(375,300+i);
```

```
        glEnd();

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

            glutSwapBuffers();
            glutPostRedisplay();
glutSwapBuffers();
            glutPostRedisplay();
            glFlush();
            glEnd();
}
glEnd();
}




void rocket_cam()
{
            count++;
count3++;

for(i=0;i<=300;i++)
{

            glClearColor(0.196078  ,0.6 ,0.8,1.0);
            glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

glColor3f(0.0,0.5,0.5);//rocket top
                 glBegin(GL_POLYGON);
                 glVertex2f(350,400+i);
                 glVertex2f(365,440+i);
                 glVertex2f(380,400+i);
                 glEnd();

        glColor3f(1.2,1.2,1.2);//rocket base
                 glBegin(GL_POLYGON);
                 glVertex2f(350,300+i);
                 glVertex2f(380,300+i);
                 glVertex2f(380,400+i);
                 glVertex2f(350, 400+i);
                 glEnd();
            glColor3f(1.0,0.0,0.0); //band color
                       glBegin(GL_POLYGON);
                       glVertex2f(350,390+i);
```

```
                glVertex2f(380,390+i);
                glVertex2f(350,380+i);
                glVertex2f(380,380+i);
                glEnd();
        glColor3f(0.8,0.4,0.0);// band color
                glBegin(GL_POLYGON);
                glVertex2f(350,370+i);
                glVertex2f(380,370+i);
                glVertex2f(350,360+i);
                glVertex2f(380,360+i);
                glEnd();
        glColor3f(1.0,0.0,0.0);//band color
                glBegin(GL_POLYGON);
                glVertex2f(350,350+i);
                glVertex2f(380,350+i);
                glVertex2f(350,340+i);
                glVertex2f(380,340+i);
                glEnd();


      glColor3f(1.0,0.0,0.0);
      glBegin(GL_POLYGON);//left_side_top
      glVertex2f(350,400+i);
      glVertex2f(330,360+i);
      glVertex2f(350,360+i);
      glEnd();
          glBegin(GL_POLYGON);//left_side_bottom
      glVertex2f(350,330+i);
      glVertex2f(330,300+i);
      glVertex2f(350,300+i);
      glEnd();
          glBegin(GL_POLYGON);//right_side_bottom
      glVertex2f(380,330+i);
      glVertex2f(400,300+i);
      glVertex2f(380,300+i);
      glEnd();
          glBegin(GL_POLYGON);//right_side_top
      glVertex2f(380,400+i);
      glVertex2f(400,360+i);
      glVertex2f(380,360+i);
      glEnd();
                glBegin(GL_POLYGON);//bottom_1_exhaust
      glVertex2f(350,300+i);
      glVertex2f(330,290+i);
      glVertex2f(365,275+i);
```

```
            glVertex2f(400,290+i);
            glVertex2f(380,300+i);
        glEnd();
            glColor3f(1.0,1.0,0.0);
                glBegin(GL_POLYGON);//bottom_2_exhaust
        glVertex2f(355,300+i);
        glVertex2f(350,290+i);
        glVertex2f(365,280+i);
        glVertex2f(380,290+i);
        glVertex2f(375,300+i);
        glEnd();

for(j=0;j<=10000000;j++)

            glutSwapBuffers();
            glutPostRedisplay();
            glFlush();
}
}

void planet()
{


            glClearColor(0.196078  ,0.6 ,0.8,1.0);
            glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

            glColor3f(0.4,0.25,0.1);
                        glBegin(GL_POLYGON);//green ground
                        glVertex2f(0.0,0.0);
                        glVertex2f(0.0,250.0);
                        glVertex2f(270.0,250.0);
                        glVertex2f(500.0,50.0);
                        glVertex2f(500.0,0.0);
                        glEnd();
                        glBegin(GL_POLYGON);//green ground
                        glVertex2f(280.0,250.0);
                        glVertex2f(500.0,250.0);
                        glVertex2f(500.0,60.0);
                        glEnd();
                        glColor3f(0.0,0.0,0.0);
                                    glBegin(GL_POLYGON);//road
                        glVertex2f(260.0,250.0);
                        glVertex2f(290.0,250.0);
                        glVertex2f(500.0,70.0);
```

```
glVertex2f(500.0,40.0);
glEnd();
glColor3f(0.0,0.0,0.0);


glColor3f(0.8,0.498039 ,0.196078);
        glBegin(GL_POLYGON);//house 1
glVertex2f(250.0,250.0);
glVertex2f(300.0,250.0);
glVertex2f(300.0,350.0);
glVertex2f(250.0,350.0);
glEnd();
glColor3f(0.7,0.7,0.7);
glBegin(GL_POLYGON);//HOUSE A
        glVertex2f(255,267.5);
        glVertex2f(275.0,267.5);
        glVertex2f(275.0,277.5);
        glVertex2f(255.0,277.5);
        glEnd();
glBegin(GL_POLYGON);//HOUSE B
        glVertex2f(255,285.0);
        glVertex2f(275.0,285);
        glVertex2f(275.0,295);
        glVertex2f(255.0,295);
        glEnd();

glBegin(GL_POLYGON);//HOUSE C
        glVertex2f(255,302.5);
        glVertex2f(275.0,302.5);
        glVertex2f(275.0,312.5);
        glVertex2f(255.0,312.5);
        glEnd();

glBegin(GL_POLYGON);//HOUSE D
        glVertex2f(255,320.0);
        glVertex2f(275.0,320.0);
        glVertex2f(275.0,330.0);
        glVertex2f(255.0,330.0);
        glEnd();

glBegin(GL_POLYGON);//HOUSE E
        glVertex2f(285,267.5);
        glVertex2f(295.0,267.5);
        glVertex2f(295.0,277.5);
        glVertex2f(285.0,277.5);
```

```
                glEnd();

        glBegin(GL_POLYGON);//HOUSE F
                glVertex2f(285,285.0);
                glVertex2f(295.0,285);
                glVertex2f(295.0,295);
                glVertex2f(285.0,295);
                glEnd();

        glBegin(GL_POLYGON);//HOUSE G
                glVertex2f(285,302.5);
                glVertex2f(295.0,302.5);
                glVertex2f(295.0,312.5);
                glVertex2f(285.0,312.5);
                glEnd();

        glBegin(GL_POLYGON);//HOUSE H
                glVertex2f(285,320.0);
                glVertex2f(295.0,320.0);
                glVertex2f(295.0,330.0);
                glVertex2f(285.0,330.0);
                glEnd();
                glColor3f(0.647059 ,0.164706  ,0.164706);
                glBegin(GL_POLYGON);//solid cone
                glVertex2f(26,250);
                glVertex2f(52,250);
                glVertex2f(39,290);
                glEnd();
                semicircle(20.0,50,300);

glColor3f(0.0,0.0 ,0.0);
                glBegin(GL_LINES);//wires
                glVertex2f(37,313);
                glVertex2f(62,310);
                glVertex2f(63,287);
                glVertex2f(62,310);
                glEnd();
        glColor3f(1.0,1.0,1.0);

        glEnd();
        glPointSize(2.0);

glColor3f(1.0,1.0 ,1.0);
                glBegin(GL_POINTS);//road paint
                glVertex2f(497,56);
```

```
                    glVertex2f(488,65);
                    glVertex2f(479,74);
                    glVertex2f(470,83);
                    glVertex2f(460,92);
                    glVertex2f(450,101);
                    glVertex2f(439,110);
                    glVertex2f(428,119);
                    glVertex2f(418,128);
                    glVertex2f(408,137);
                    glVertex2f(398,146);
                    glVertex2f(388,155);
                    glVertex2f(378,164);
                    glVertex2f(366,173);
                    glVertex2f(356,182);
                    glVertex2f(346,191);
                    glVertex2f(336,200);
                    glVertex2f(324,209);
                    glVertex2f(314,218);
                    glVertex2f(304,227);
                    glVertex2f(294,234);
                    glVertex2f(284,243);
            glVertex2f(278,248);

                    glEnd();

        glColor3f(0.8,0.498039 ,0.196078);
        glBegin(GL_POLYGON);//core
                glVertex2f(237.5,20.0);
                glVertex2f(262.5,20.0);
                glVertex2f(262.5,120.0);
                glVertex2f(237.5,120.0);


        glEnd();

        glColor3f(1.0,1.0,1.0);//bonnet
        glBegin(GL_POLYGON);//front
        glVertex2f(237.5,120.0);
        glVertex2f(262.5,120.0);
        glVertex2f(250,170.0);
        glEnd();
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_POLYGON);//left_side_top
        glVertex2f(237.5,120.0);
        glVertex2f(217.5,95.0);
```

```
glVertex2f(237.5,95.0);
glEnd();
        glBegin(GL_POLYGON);//left_side_bottom
glVertex2f(237.5,20.0);
glVertex2f(217.5,20.0);
glVertex2f(237.5,70.0);
glEnd();
        glBegin(GL_POLYGON);//right_side_bottom
glVertex2f(262.5,20.0);
glVertex2f(282.5,20.0);
glVertex2f(262.5,70.0);
glEnd();
        glBegin(GL_POLYGON);//right_side_top
glVertex2f(262.5,120.0);
glVertex2f(262.5,95.0);
glVertex2f(282.5,95.0);
glEnd();
glColor3f(0.556863 ,0.137255  ,0.419608);
        glBegin(GL_POLYGON);//bottom_1_exhaust
glVertex2f(237.5,20.0);
glVertex2f(244.5,20.0);
glVertex2f(241,0.0);
glEnd();
        glBegin(GL_POLYGON);//bottom_2_exhaust
glVertex2f(246.5,20.0);
glVertex2f(253.5,20.0);
glVertex2f(249.5,0.0);
glEnd();
        glBegin(GL_POLYGON);//bottom_3_exhaust
glVertex2f(262.5,20.0);
glVertex2f(255.5,20.0);
glVertex2f(258.5,0.0);
glEnd();

glBegin(GL_POLYGON);//left_stand_holder
glVertex2f(182.5,85.0);
glVertex2f(182.5,0.0);
glVertex2f(187.5,0.0);
glVertex2f(187.5,80.0);
glVertex2f(237.5,80.0);
glVertex2f(237.5,85.0);
glVertex2f(182.5,85.0);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(312.5,85.0);//right_stand_holder
```

```
        glVertex2f(312.5,0.0);
        glVertex2f(307.5,0.0);
        glVertex2f(307.5,80.0);
        glVertex2f(262.5,80.0);
        glVertex2f(262.5,85.0);
        glVertex2f(312.5,85.0);
        glEnd();

        for(j=0;j<=1000000;j++)
                ;
        glutSwapBuffers();
        glutPostRedisplay();
        glFlush();
}

void stars()
{

        glColor3f(1.0,1.0,1.0);
        glPointSize(1.5);
        glBegin(GL_POINTS);
        glVertex2i(10,20);
        glVertex2i(20,100);
        glVertex2i(30,10);
        glVertex2i(15,150);
        glVertex2i(17,80);
        glVertex2i(200,200);
        glVertex2i(55,33);
        glVertex2i(400,300);
        glVertex2i(330,110);
        glVertex2i(125,63);
        glVertex2i(63,125);
        glVertex2i(20,10);
        glVertex2i(110,330);
        glVertex2i(440,430);
        glVertex2i(32,65);
        glVertex2i(110,440);
        glVertex2i(210,230);
        glVertex2i(390,490);
        glVertex2i(12,90);
        glVertex2i(400,322);
        glVertex2i(420,366);
        glVertex2i(455,400);
        glVertex2i(20,20);
        glVertex2i(111,120);
        glVertex2i(401,200);
        glVertex2i(230,30);
        glVertex2i(220,20);
        glVertex2i(122,378);
        glVertex2i(133,340);
        glVertex2i(345,420);
        glVertex2i(130,360);
        glVertex2i(333,120);
        glVertex2i(250,22);
        glVertex2i(242,11);
        glVertex2i(280,332);
        glVertex2i(233,40);
        glVertex2i(210,418);
        glVertex2i(256,12);
        glVertex2i(288,232);
        glVertex2i(247,36);
        glVertex2i(229,342);
        glVertex2i(257,47);
        glVertex2i(290,63);
        glVertex2i(232,72);
        glVertex2i(243,143);
        glVertex2i(100,200);
        glVertex2i(90,250);
        glVertex2i(80,225);
        glVertex2i(50,333);
        glVertex2i(60,350);
        glVertex2i(243,143);
        glVertex2i(243,143);
        glEnd();
}

void stars1()
{
        int l;
        glColor3f(1.0,1.0,1.0);
        glPointSize(1.5);
        glBegin(GL_POINTS);
        glVertex2i(50,20);
        glVertex2i(70,100);
        glVertex2i(80,10);
        glVertex2i(65,150);
        glVertex2i(67,80);
        glVertex2i(105,33);
        glVertex2i(450,300);
        glVertex2i(380,110);
```

```
glVertex2i(175,63);
glVertex2i(113,125);
glVertex2i(70,10);
glVertex2i(160,330);
glVertex2i(490,430);
glVertex2i(82,65);
glVertex2i(160,440);
glVertex2i(440,490);
glVertex2i(62,90);
glVertex2i(450,322);
glVertex2i(420,366);
glVertex2i(455,400);
glVertex2i(60,20);
glVertex2i(111,120);
glVertex2i(451,200);
glVertex2i(280,30);
glVertex2i(220,20);
glVertex2i(132,378);
glVertex2i(173,340);
glVertex2i(325,420);
glVertex2i(180,360);
glVertex2i(383,120);
glVertex2i(200,22);
glVertex2i(342,11);
glVertex2i(330,332);
glVertex2i(283,40);
glVertex2i(210,418);
glVertex2i(256,12);
glVertex2i(288,232);
glVertex2i(247,36);
glVertex2i(229,342);
glVertex2i(257,47);
glVertex2i(290,63);
glVertex2i(232,72);
glVertex2i(243,143);
glVertex2i(100,200);
glVertex2i(90,250);
glVertex2i(80,225);
glVertex2i(50,333);
glVertex2i(60,350);
glVertex2i(243,143);
glVertex2i(243,143);
glEnd();
for(l=0;l<=100000;l++)
        ;
```

```
}


void display1()
{

count1++;
 if(count1==460)
      flag=1;
   if(flag==0)
       staticrocket();
 else if((count1==151)| (count1==152))
       rocket_cam();
  else{
      motionrocket();
        planet();
        glEnd();
      }

}


void myinit()

    {

        glClearColor(0.2,0.5 ,1.0,0.0);
        glPointSize(5.0);
        gluOrtho2D(0.0,600.0,0.0,600.0);
    }

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

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(900,900);
        glutInitWindowPosition(-5.0,5.0);
        glutCreateWindow("PSLV -C 37");
        glutDisplayFunc(display1);
        myinit();
        glutMainLoop();
            return 0;
    }
```

7. SAMPLE OUTPUT

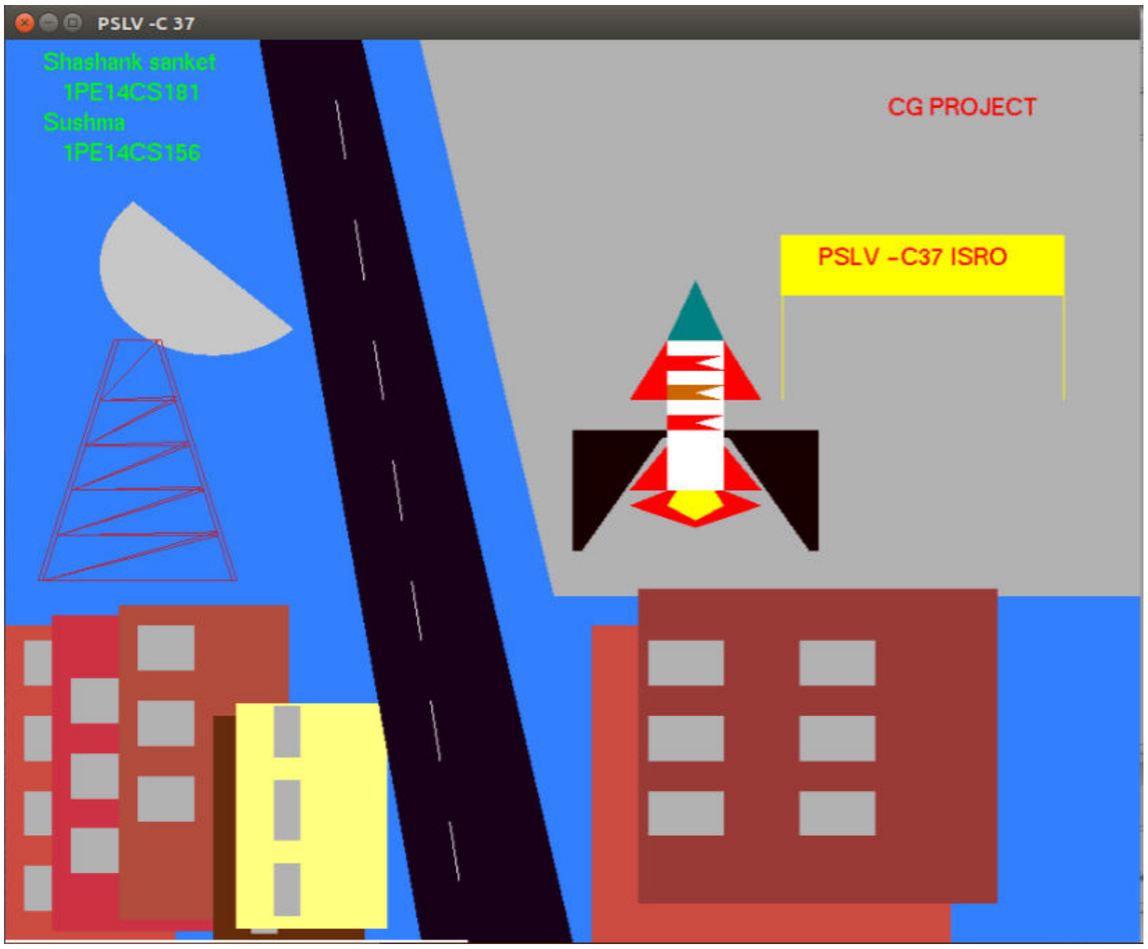

fig 7.1: Static Ground Screen simulation (stage 1).

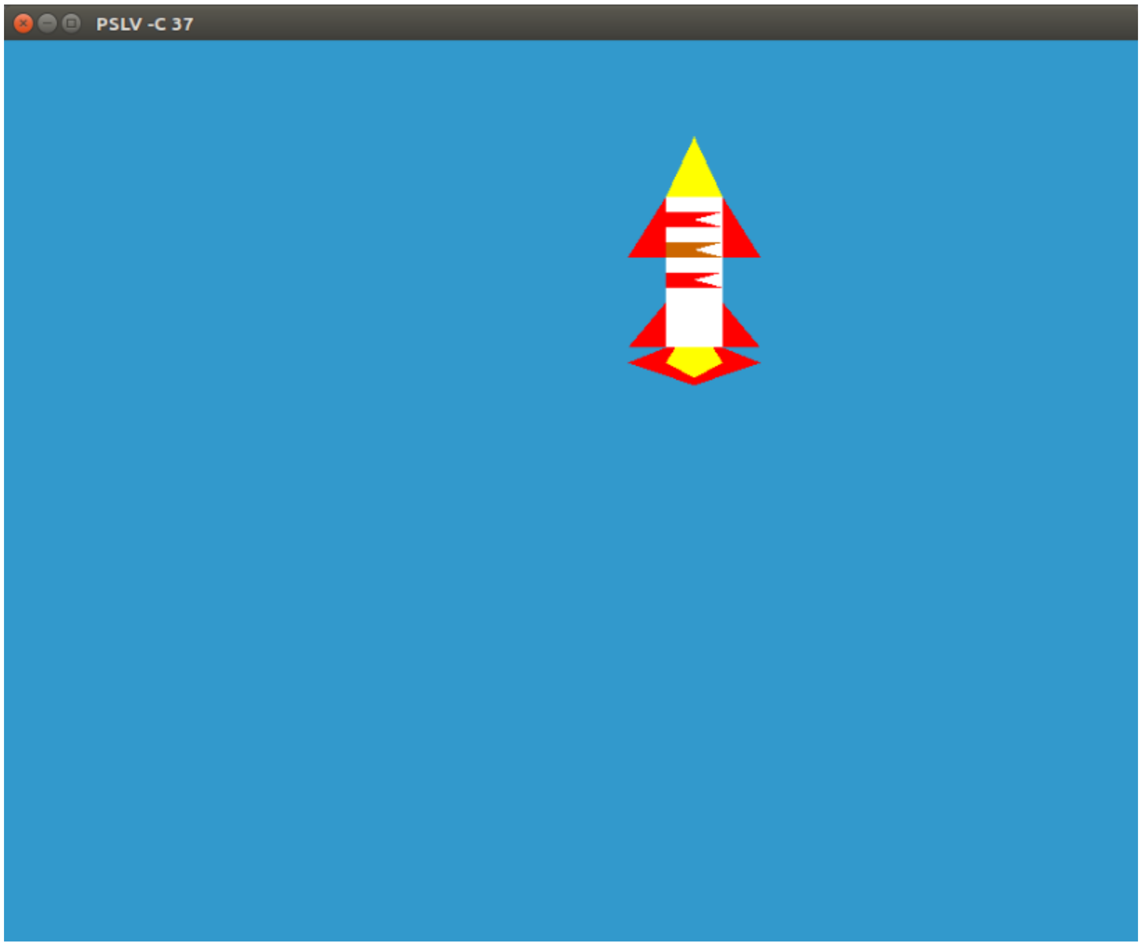

fig 7.2: Second screen of Simulation when rocket is launched (stage 2).

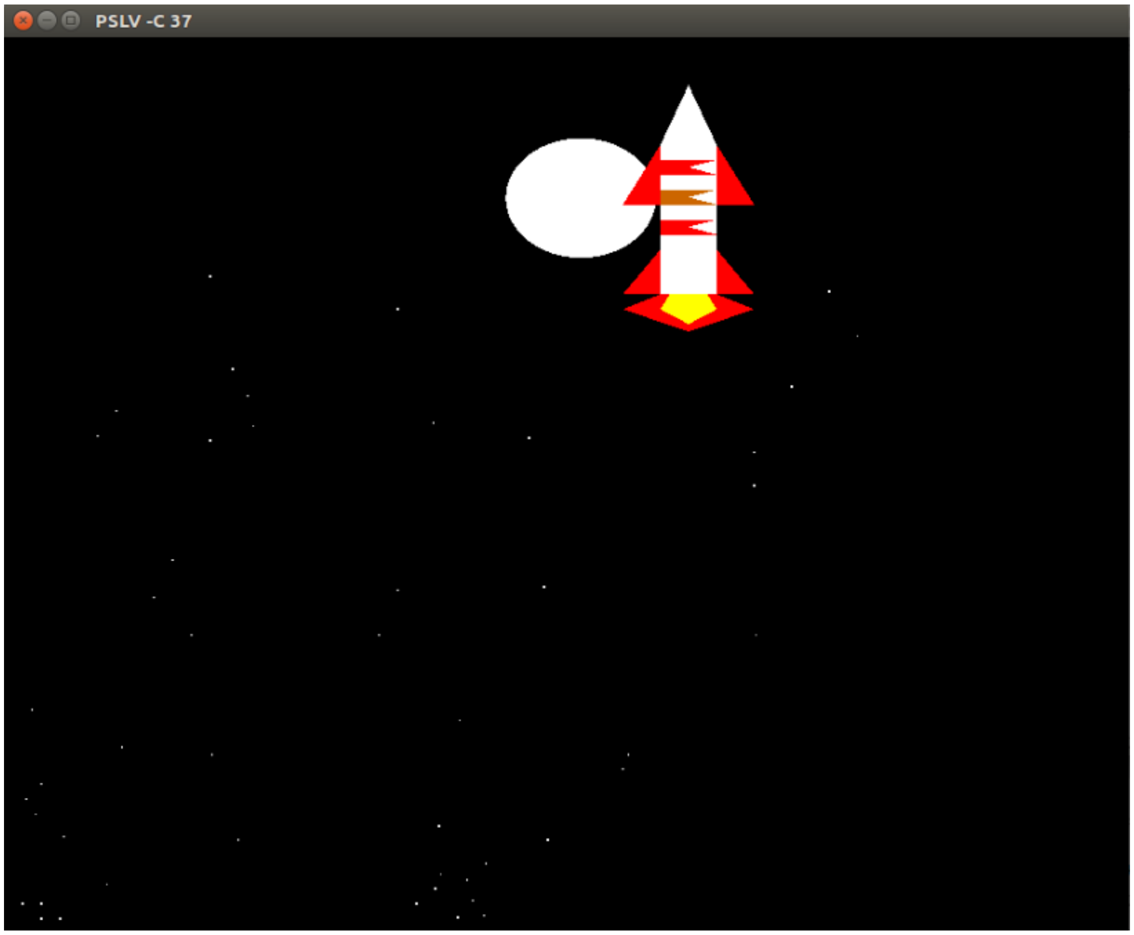

fig 7.3:Third animated screen showing Moon pass by shuttle from Top to Left (stage 3).

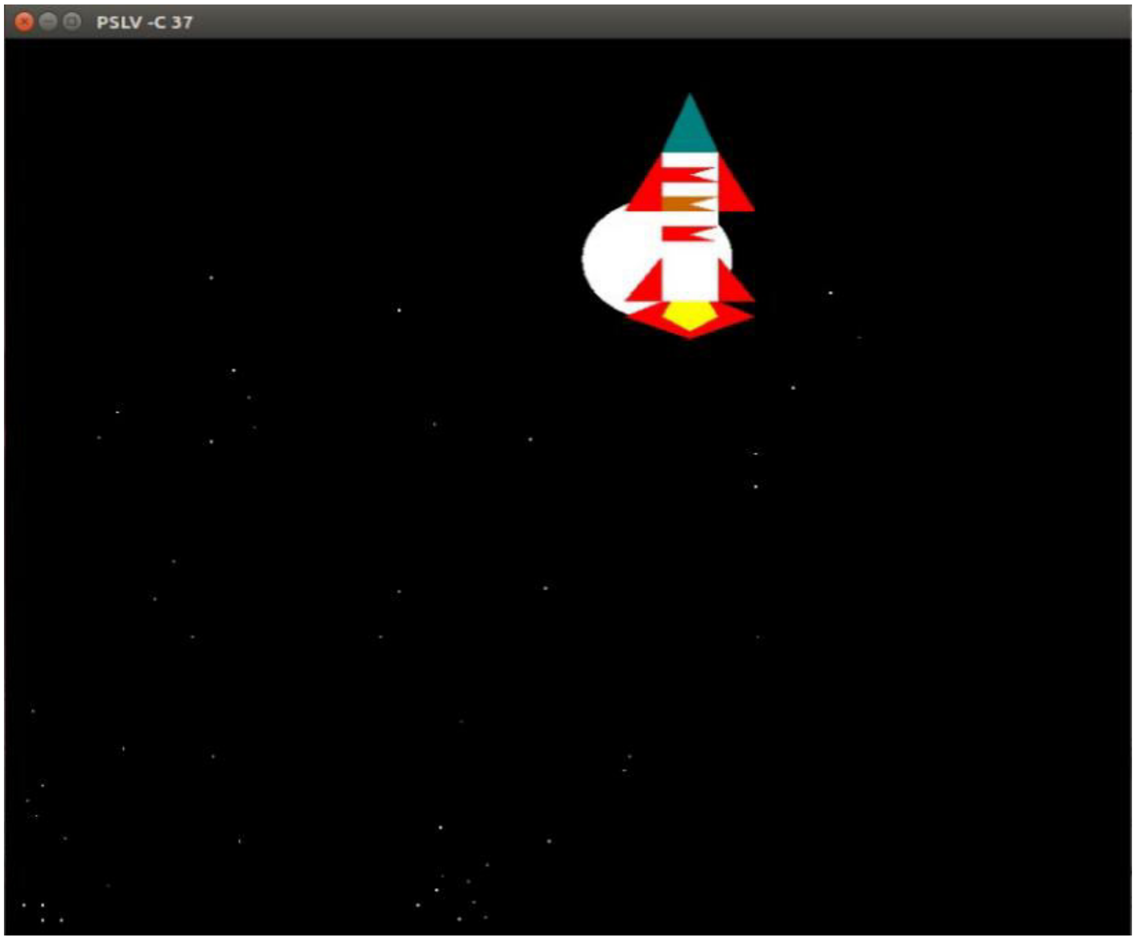

fig 7.4: Simulation of shuttle pass by Moon (stage 3).

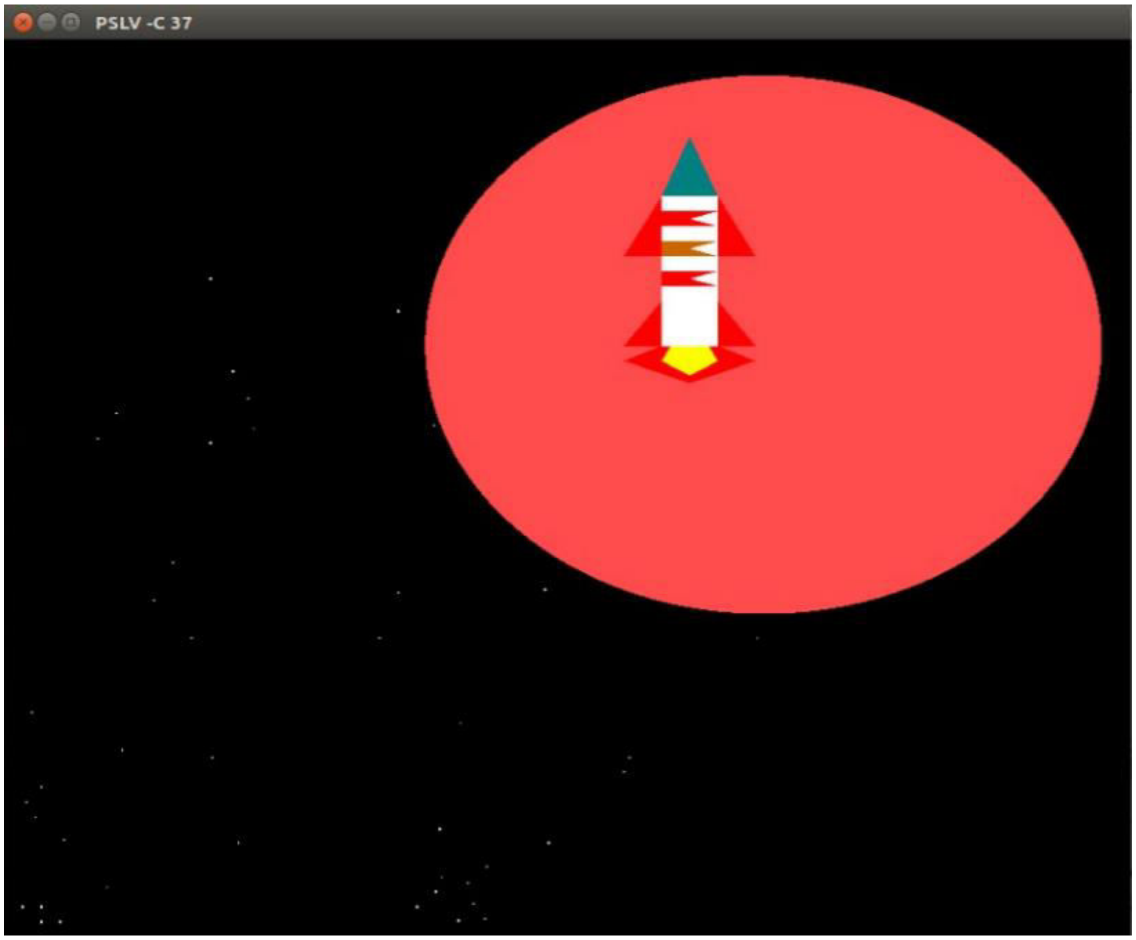

fig 7.5:Last and the Final stage where shuttle reaches Mars and Simulation Ends in Mars. (Stage 4).

8. CONCLUSIONS AND FUTURE SCOPE

In this simple graphics in C we use the logic of launching a Space Shuttle, its journey into space crossing the Mars. Different stages of Rocket propulsion is avoided due to its complication. This computer graphics program in C, simply demonstrate the Space Shuttle launch, it's way to leave the earth entering space and journey into the space.

The logic behind the Journey of Space Shuttle or Rocket computer graphics program in C is little complicated but easy to understand. All the objects created quite easily with primitive functions, which is easy to code. While there are two type of rocket - static and moveable. Separate functions has been defined for them.

The most important code is the programming of fume, the rocket burning fumes. Without it the whole graphics program will look dull.

FUTURE SCOPE

It would be tough to have small flag of India on the rocket but you can write India in letter on the rocket. The rocket propulsion has many stages that can also be added in future to the **computer graphics programs in C.** One key issue with the program is that it's not self-terminated, that means in space it will run till program is close. Manual closing can be remove with self-close.

9. BIBLIOGRAPHY

1. Interactive Computer Graphics A Top-Down Approach with OpenGL - Edward Angel, 5th Edition, Addison-Wesley, 2008.

2. Computer Graphics Using OpenGL - F.S. Hill, Jr. 2nd Edition, Pearson Education, 2001.

3. OpenGL Programming Guide or RED BOOK.

4. https:// www.opengl.org