VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590 014

A MINI PROJECT REPORT ON

"WORKING OF SATELLITE"

Submitted in partial fulfillment of the requirements for

CG&V MINI PROJECT [18CSL67] IN

6th SEM COMPUTER SCIENCE AND ENGINEERING 2020-2021

Submitted by:

K Ganesh	1GG18CS023
Pruthvi Kumar U	1GG18CS032
Sagar Gowda B V	1GG18CS034

UNDER THE GUIDANCE OF

Prof. Sri Ramachandra L

Associate Professor Dept. of CS&E

GEC, Ramanagaram.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

GOVERNMENT ENGINEERING COLLEGE

B.M. ROAD, RAMANAGARA-562 159

GOVERNMENT ENGINEERING COLLEGE

Ramanagara-562159

Department of Computer Science & Engineering

CERTIFICATE

Certified that the project entitled " **WORKING OF SATTELITE** " carried out by **"K Ganesh , Pruthvi Kumar U , Sagar Gowda B V "** Submitted by the partial fulfillment of the requirements for the award of the degree of COMPUTER GRAPHICS LABORATORY **[18CSL67] with Mini Project in 6th semester Computer Science and Engineering** of **Visvesvaraya Technological University**, Belagavi during the year 2020-2021. The Project report has approved as it satisfies the academic requirements in respect to the Mini Project Work prescribed for the **Bachelor of Engineering Degree.**

Prof. Sri Ramachandra L	**Prof. Dr. Vasanth G**
[Guide, Dept of CSE]	[HOD Dept of CSE]

Name of the Examiners	Signature with Date
1……………………….	……………………
2……………………...	…………………….

ACKNOWLEDGEMENT

We consider it a privilege to whole-heartedly express my gratitude and respect to everyone who guided and helped me in the successful completion of this Project Report.

We are very thankful to the principal **Dr. G.PUNDARIKA** for being kind enough to provide me an opportunity to work on a Project in this institution.

We are also thankful to **Prof. Dr. Vasanth G** , HOD, Department of Computer Science, for her co-operation and encouragement at all moments of my approach.

We would greatly mention the enthusiastic influence provided by

Prof. Sri Ramachandra L, Associate Professor, as our Project Guide, for her ideas and co- operation showed on us during my venture and making this Project a great success.

We would also like to thank my parents and well-wishers as well as my dear classmates for their guidance and their kind co-operation.

Finally, it is my pleasure and happiness to the friendly co-operation showed by all the staff members of Computer Science Department, GECR.

CONTENTS

Abstract

Main aim of this Mini Project is to illustrate the concepts of working of a Satellite in OpenGL. A Satellite is an object which has been placed into orbit by human endeavor. Such objects are sometimes called artificial satellites to distinguish them from natural satellites such as the Moon.

Satellites are used for many purposes. Common types include military and civilian Earth observation satellites, communications satellites, navigation satellites, weather satellites, and research satellites. This pushed the entire network into a 'congestion collapse' where most packets were lost and the resultant throughput was negligible.

We have used input devices like mouse and key board to interact with program. We have also used Solid Cube for forming a complete network setup which help to understand concept of Congestion Control very well. To differentiate between objects, we have used different colors for different objects. We have added menu which makes the program more interactive.

In this project we have used a small Solid Cube to represent a data, which travels as data transfer from source to destination. We have used font family for indicating the name of objects as we can see in this project

System specifications

- **SOFTWARE REQUIREMENTS :**
  - MICROSOFT VISUAL C++
  - OPENGL

- **HARDWARE REQUIREMENT :**
  - GRAPHICS SYSTEM
  - Pentium P4 with 256 of Ram(Min)

Introduction to 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 or pixels.

OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

Primitives and Commands

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes.

You can control modes independently of each other; that is, setting one mode does not affect whether other modes are set .Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

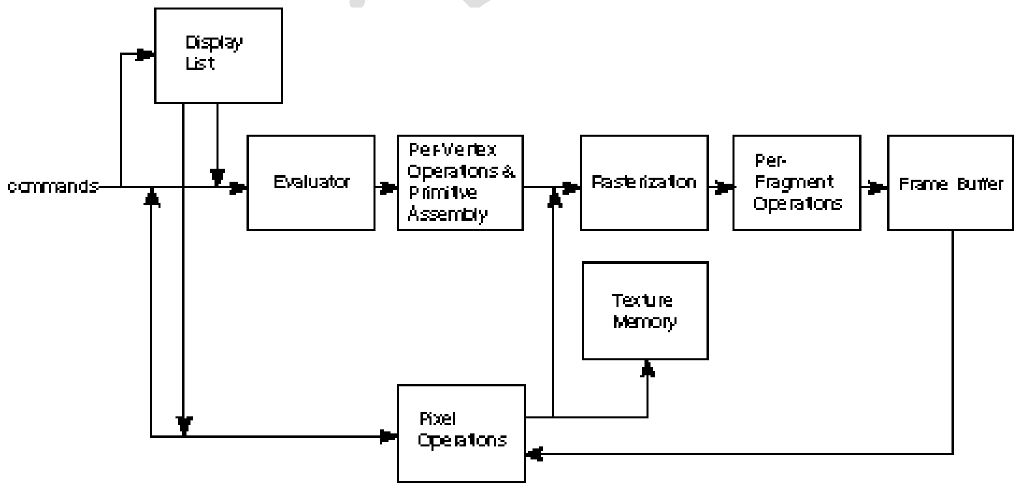
Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The type of clipping depends on which primitive the group of vertices represents.

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 is consistent with complete execution of all previously issued OpenGL commands.

Basic OpenGL Operation

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

Figure . OpenGL Block Diagram



As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing later.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

Each fragment so produced is fed into the last stage,

per-fragment operations, which performs the final operations on the data before it is stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-value s(for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

Implementation

This program is implemented using various openGL functions which are shown below.

Various functions used in this program.

- glutInit() : interaction between the windowing system and OPENGL is initiated.
- glutInitDisplayMode() : used when double buffering is required and depth information is required
- glutCreateWindow() : this opens the OPENGL window and displays the title at top of the window
- glutInitWindowSize() : specifies the size of the window
- glutInitWindowPosition() : specifies the position of the window in screen co-ordinates
- glutKeyboardFunc() : handles normal ascii symbols
- glutSpecialFunc() : handles special keyboard keys
- glutReshapeFunc() : sets up the callback function for reshaping the window
- glutIdleFunc() : this handles the processing of the background
- glutDisplayFunc() : this handle redrawing of the window

- glutMainLoop() : this starts the main loop, it never returns
- glViewport() : used to set up the viewport
- glVertex3fv() : used to set up the points or vertices in three dimensions
- glColor3fv() : used to render color to faces
- glFlush() : used to flush the pipeline
- glutPostRedisplay() : used to trigger an automatic redraw of the object
- glMatrixMode() : used to set up the required mode of the matrix
- glLoadIdentity() : used to load or initialize to the identity matrix

- glTranslatef() : used to translate or move the rotation center from one point to another in three dimensions.
- glRotatef() : used to rotate an object through a specified rotation angle

Interaction with program

This program includes interaction through keyboard.

- S -> Start the Project
- t/T -> to transmit and receive signals
- Q-> Quit

Source code

```
#include <windows.h>
#include<string.h>
#include<stdarg.h>
#include<stdio.h>
#include <GL/glut.h>
#include <math.h>
static double x=0.0;
static double move=-60;
static float rx[100]={0}, ry[100]={0};
//control waves
static double w1=0,w2=0,w3=0;
static bool transmit=false;
void *font;
void *currentfont;
void setFont(void *font)
{
currentfont=font;
}
void drawstring(float x,float y,float z,char *string)
{
char *c;
```

```
glRasterPos3f(x,y,z);


for(c=string;*c!='\0';c++)
{ glColor3f(0.0,1.0,1.0);
glutBitmapCharacter(currentfont,*c);
}
}
void
stroke_output(GLfloat x, GLfloat y, char *format,...)
{
va_list args;
char buffer[200], *p;
va_start(args, format);
vsprintf(buffer, format, args);
va_end(args);
glPushMatrix();
glTranslatef(-2.5, y, 0);
glScaled(0.003, 0.005, 0.005);
for (p = buffer; *p; p++)
 glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
glPopMatrix();
}


void satellite(){
glRotatef(60,1,0,0);
//body
glPushMatrix();
glColor3f(0.2,0.2,0.2);
glScaled(1,0.6,1);
glTranslatef(3.0,0,0.0);
glutSolidCube(0.4);
glPopMatrix();
//Solar Panels
```

```
glPushMatrix();
glColor3f(0.3,0.3,0.3);
glTranslatef(3,0,0.0);
//glRotatef(45,1,0,0);
glScaled(3.7,0.0,1);
glutSolidCube(0.4);
glPopMatrix();


glPushMatrix();
glColor3f(0.2,0.1,0.1);
glTranslatef(3.0,0,-0.4);
glScaled(0.5,0.5,0.5);
glutSolidSphere(0.3,50,50);
glPopMatrix();
glPushMatrix();
glColor3f(0.2,0.2,0.1);
glTranslatef(3.0,0,0.4);
glScaled(0.4,0.4,0.3);
glutSolidTorus(0.3,0.2,20,20);
glPopMatrix();
}
// Second Screen
void sat2(double ang)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-13.0f);


 glRotatef(ang,0.0f,1.0f,0.0f);
//earth
glPushMatrix();
glColor3f(0.3,0.6,1);
//glScaled(0.8,0.04,0.8);
```

```
//glTranslatef(0.0,0,0.0);
glutSolidSphere(2.0,50,50);
glPopMatrix();
satellite();

 glFlush();
 glutSwapBuffers();
}
void building(float x1,float y1,float z1){
//Main Structure


glPushMatrix();
glColor3f(0.5,0.5,0.5);
glTranslatef(x1,y1,z1);
glScaled(0.5,1.5,0.5);
glutSolidCube(2);
glPopMatrix();
//Dish on top
glPushMatrix();
glColor3f(1,1,0);
glTranslatef(x1,y1+1.8,z1);
glRotatef(60,1,0,0);
glScaled(0.5,1.5,0.5);
glutSolidCone(0.5,1,20,20);
glPopMatrix();
//windows
glPushMatrix();
glColor3f(0.1,0,0);
glTranslatef(x1-0.2,y1+0.7,z1);
glScaled(0.5,0.5,0.5);
//glutSolidCube(.3);
for(float j=-3;j<1.5;j+=.8)
```

```
{
for(float i=0;i<1;i+=0.8)
{
glPushMatrix();
glTranslatef(i,j,1);
glutSolidCube(0.4);
glPopMatrix();
}
}
glPopMatrix();
}
void waves(){
glPushMatrix();
glTranslatef(0,1,0);
glScaled(0.05,0.5,0.1);
glutSolidCube(0.5);
glPopMatrix();


glPushMatrix();
glRotatef(-8,0,0,1);
glTranslatef(0.01,1,0);
glScaled(0.05,0.5,0.1);
glutSolidCube(0.5);
glPopMatrix();
glPushMatrix();
glRotatef(8,0,0,1);
glTranslatef(-0.01,1,0);
glScaled(0.05,0.6,0.1);
glutSolidCube(0.5);
glPopMatrix();
}
void sat1(){
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-13.0f);


 //glRotatef(x,0.0f,1.0f,0.0f);
//Moon
glPushMatrix();
glColor3f(1,1,1);
glTranslatef(-3.8,2.8,0);
glScaled(0.5,0.5,0.1);
glutSolidSphere(0.6,50,50);
glPopMatrix();
//Earth
glPushMatrix();
glColor3f(0.2,0.2,1);
glTranslatef(0,-12,0);
//glScaled(0.8,0.04,0.8);
glutSolidSphere(10.0,50,50);
glPopMatrix();
//Building Center
glPushMatrix();
glColor3f(0,1,1);
glRotatef(10,1,0,0);
building(1.2,-1.2,3.2);


glPopMatrix();
//Building left
glPushMatrix();
glColor3f(0,1,1);
glRotatef(5,0,0,1);
building(-3.8,-1.2,0);
glPopMatrix();
//signal
glPushMatrix();
glColor3f(0,0,1);
```

```
if(transmit){
glRotatef(-25,0,0,1);
glTranslatef(-1.25,-1.6+w1,0);
}else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//Main Dish


//Tower
glPushMatrix();
glColor3f(1,1,1);
glTranslatef(-1,-2,4);
glRotatef(270,1,0,0);
glScaled(1.0,1,2.0);
glutWireCone(0.5,1,4,10);
glPopMatrix();
//Dish
glPushMatrix();
glColor3f(1,1,1);
glTranslatef(-1.08,0.2,3);
glRotatef(60,1,0,0);
glScaled(0.7,1.3,0.7);
glutSolidCone(0.4,0.5,20,20);
glPopMatrix();
//Building right
glPushMatrix();
glColor3f(0,1,1);
glRotatef(-5,0,0,1);


building(3.8,-1.2,0);
glPopMatrix();
//Saltellite
glPushMatrix();
```

```
glTranslatef(-3,3.0,0);
satellite();
glPopMatrix();
//Ack to right building
glPushMatrix();
if(transmit){
glRotatef(50,0,0,1);
glTranslatef(2.8,3.2-w2,0);
}else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//Ack to Left building


glPushMatrix();
if(transmit){
glRotatef(-50,0,0,1);
glTranslatef(-2.8,3.2-w2,0);
}else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//Ack to Center building
glPushMatrix();
if(transmit){
glRotatef(23,0,0,1);
glTranslatef(1,3.2-w3,3.3);
}
else glTranslatef(1,20,3.3);
waves();
glPopMatrix();
//stars


glPointSize(5);
for(int j=0;j<100;j++)
```

```
{
for(int i=0;i<100;i++)
{
 rx[j]=rand()/500;
 ry[i]=rand()/500;
glBegin(GL_POINTS);
glColor3f(0,2,2);
glVertex3f(-6+rx[j],ry[i],-5);
glEnd();
}
}
glPushMatrix();
//glScaled(1.1,2.0,0.1);
glTranslatef(0.0,0.0,-2.0);
setFont(GLUT_BITMAP_TIMES_ROMAN_24);


glColor3f(1,1,1);
drawstring(1,3.7,-1.0,"Satelitte");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,1);
drawstring(-4.4,.5,-1.0,"Reciever");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,0);
drawstring(0,-2,7,"Reciever");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,1);
drawstring(-1.5,-1,-1.0,"Transmitter");
setFont(GLUT_BITMAP_TIMES_ROMAN_24);
glColor3f(1,1,1);
drawstring(3.2,1,3,"Reciever");
glPopMatrix();
glFlush();
 glutSwapBuffers();
}
```

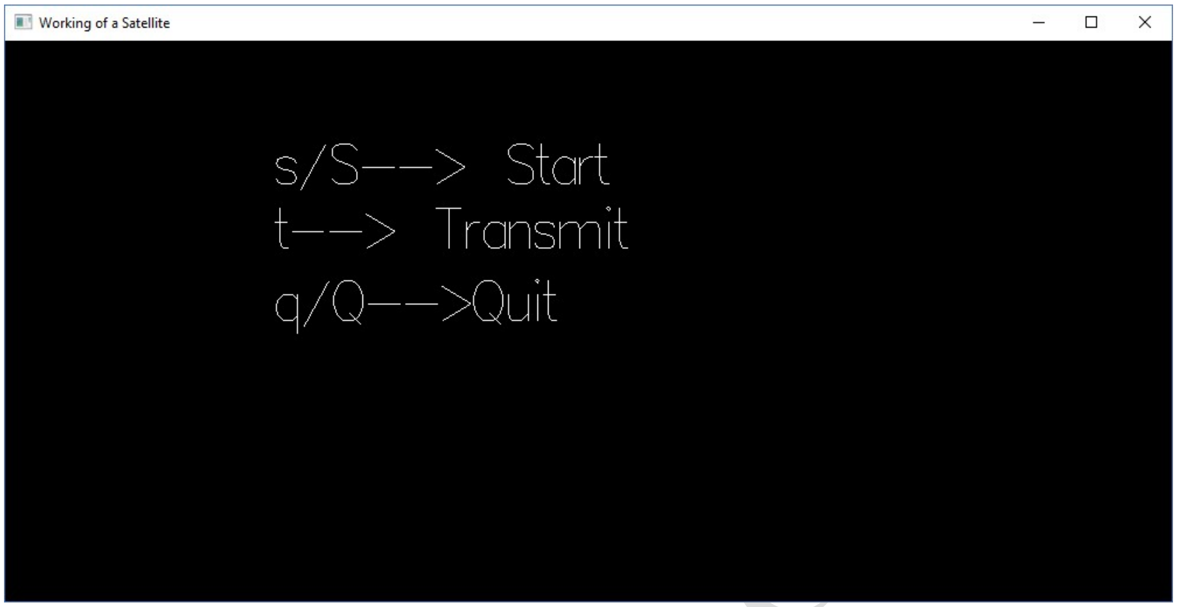
```
// Third Screen
void sat3(double ang)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(0.0f,0.0f,-13.0f);
 glRotatef(ang,0.0f,1.0f,0.0f);
//earth
glPushMatrix();
glColor3f(0.3,0.6,1);
//glScaled(0.8,0.04,0.8);
//glTranslatef(0.0,0,0.0);
glutSolidSphere(2.0,50,50);
glPopMatrix();
satellite();

 glFlush();
 glutSwapBuffers();
}


void e()
{
x-=0.07;
sat2(x);
 }
void s()
{
x-=0.07;
sat2(x);
 }
void S()
{
```

```
x += .07;


if(transmit)
{
if(w1<=4.2)
w1+=0.01;
if(w1>=2.5 && w2<=6.9)
w2+=0.01;
if(w1>=2.5 && w3<=5)
w3+=0.01;
}
sat1();
}
void doInit()
{
 /* Background and foreground color */
 glClearColor(0.0,0.0,0.0,0);
 glViewport(0,0,640,480);


/* Select the projection matrix and reset it then
 setup our view perspective */
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluPerspective(30.0f,(GLfloat)640/(GLfloat)480,0.1f,200.0f);
 /* Select the modelview matrix, which we alter with rotatef() */
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glClearDepth(2.0f);
 glEnable(GL_DEPTH_TEST);
glEnable( GL_COLOR_MATERIAL );
 glDepthFunc(GL_LEQUAL);
}
void display()
```

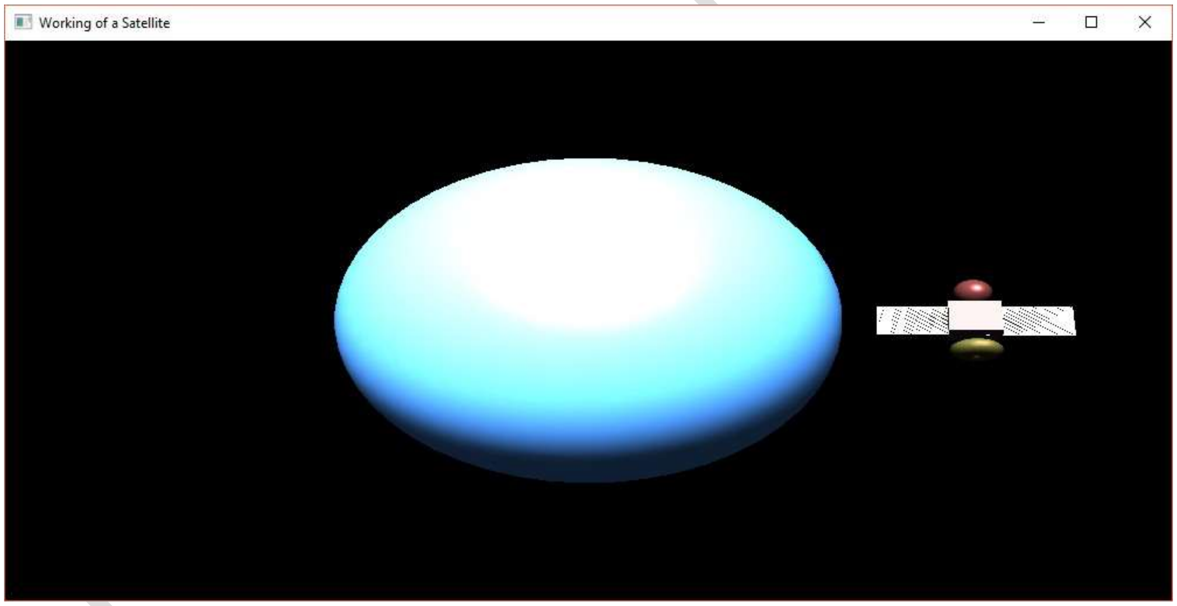
```
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glTranslatef(0.0f,0.0f,-13.0f);


stroke_output(-2.0, 1.7, "s/S--> Start");
stroke_output(-2.0, 0.9, "t--> Transmit");
stroke_output(-2.0, 0.0, "q/Q-->Quit");
GLfloat mat_ambient[]={0.0f,1.0f,2.0f,1.0f};
GLfloat mat_diffuse[]={0.0f,1.5f,.5f,1.0f};
GLfloat mat_specular[]={5.0f,1.0f,1.0f,1.0f};
GLfloat mat_shininess[]={50.0f};
glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
GLfloat lightIntensity[]={1.7f,1.7f,1.7f,1.0f};
GLfloat light_position3[]={0.0f,8.0f,10.0f,0.0f};
glLightfv(GL_LIGHT0,GL_POSITION,light_position3);
glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);
GLfloat lightIntensity1[]={1.7f,1.7f,1.7f,1.0f};
GLfloat light_position31[]={-2.0f,8.0f,10.0f,0.0f};


glLightfv(GL_LIGHT1,GL_POSITION,light_position31);
glLightfv(GL_LIGHT1,GL_DIFFUSE,lightIntensity1);
glEnable(GL_COLOR_MATERIAL);
glFlush();
glutSwapBuffers();
 }
void menu(int id)
{
switch(id)
{
```

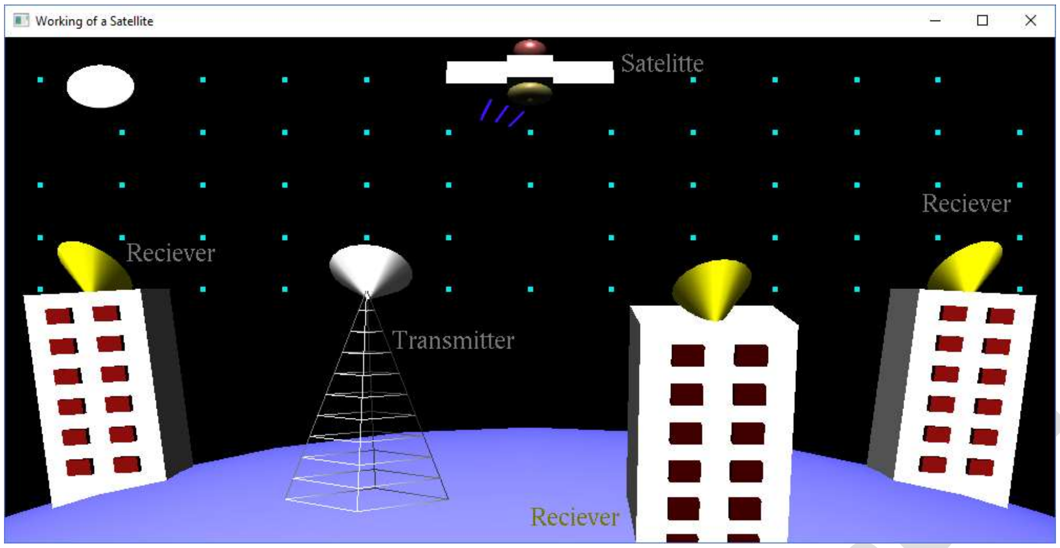
```
case 1:glutIdleFunc(S);
break;
case 2:glutIdleFunc(s);
break;
case 5:exit(0);
break;


}
glFlush();
glutSwapBuffers();
glutPostRedisplay();
}
void mykey(unsigned char key,int x,int y)
{
if(key=='s')
{
glutIdleFunc(s);
}
if(key=='S')
{
glutIdleFunc(S);
}
if(key=='e')
{
glutIdleFunc(e);
}


if(key=='t')
{ transmit=!transmit;
if(!transmit)
{
w1=0;
w2=0;
w3=0;
}
glutIdleFunc(S);
}
if(key=='q'||key=='Q')
{
exit(0);
}
}

int main(int argc, char *argv[])
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
 glutInitWindowSize(1000,480);
 glutInitWindowPosition(0,0);
 glutCreateWindow("Working of a Satellite");
 glutDisplayFunc(display);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glEnable(GL_NORMALIZE);
glutKeyboardFunc(mykey);
glutCreateMenu(menu);
 glutAddMenuEntry("Pyramid 's'",1);
glutAddMenuEntry("Reverse Pyramid 'S'",2);
glutAddMenuEntry("Quit 'q'",5);
glutAttachMenu(GLUT_RIGHT_BUTTON);
doInit();
 glutMainLoop();

return 0;
}
```

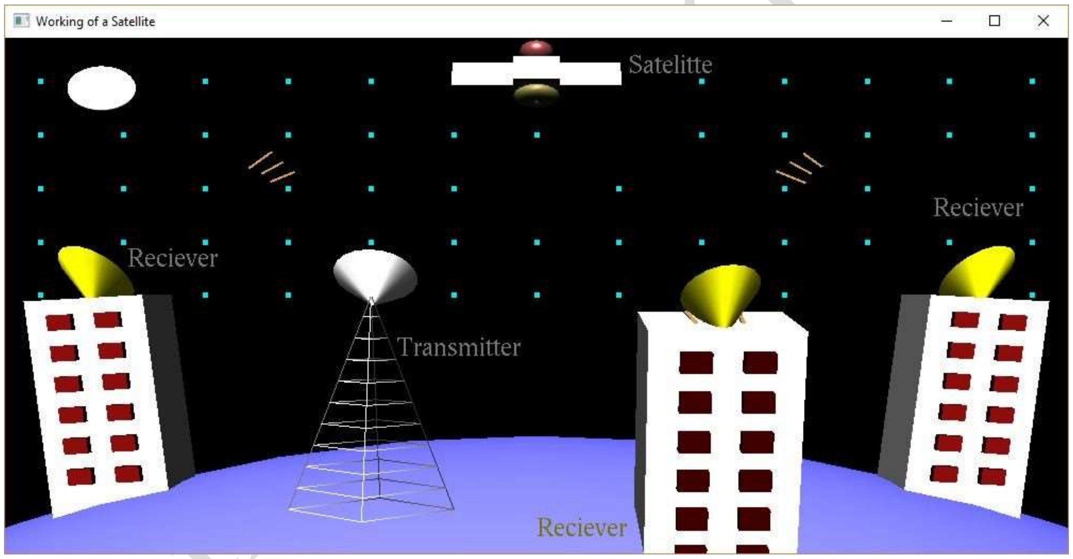
OUTPUT OF THE PROGRAM



```
s/S---> Start
t---> Transmit
q/Q--->Quit
```




Working of a Satellite
Satelitte
Reciever
Reciever
Transmitter
Reciever


Working of a Satellite
Satelitte
Reciever
Reciever
Transmitter
Reciever

Conclusions

The project "Working of a Satellite" demonstrates how signals are transmitted and received to and from a satellite.

Finally, we conclude that this program clearly illustrates the working of a satellite using OpenGL and has been completed successfully and is ready to be demonstrated.

Bibliography

WE HAVE OBTAINED INFORMATION FROM MANY RESOURCES TO DESIGN AND IMPLEMENT OUR PROJECT SUCCESSIVELY. WE HAVE ACQUIRED MOST OF THE KNOWLEDGE FROM RELATED WEBSITES. THE FOLLOWING ARE SOME OF THE RESOURCES :

TEXT BOOKS :

- INTERACTIVE COMPUTER GRAPHICS A TOP-DOWN APPROACH

  -By Edward Angel.

- COMPUTER GRAPHICS,PRINCIPLES & PRACTICES

  - Foley van dam and

  - Feiner hughes

WEB REFERENCES

http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson3.php

http://google.com

http://opengl.org