#include <windows.h>

#include <GL/glut.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#define PI 3.14159265

#define bool int

#define true 1

#define false 0

static GLfloat lpos[] = { 0.0, 0.0, 4.0, 1.0 };

static GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 };

static GLfloat black[] = { 0.0, 0.0, 0.0, 1.0 };

static GLfloat red[] = { 1.0, 0.0, 0.0, 1.0 };

static GLfloat yellow[] = { 1.0, 1.0, 0.0, 1.0 };

static float alpha = 0.0;

static float beta = PI / 6.0;

static float zoom = 10.0;

static bool lightSource = true;

float twistConstant = 0;

float rotateConstant = 0;

float numberOfObj = 1;

float numberOfTriangles = 1;

float numberOfCubes=1;

static GLdouble cpos[3];

bool showNormalVectors = false;

float x;

float y;

float z;

float twistAng;

float twistCos;

float twistSin;

float yTwisted;

float zTwisted;

void text(int x,int y,char \*s)

{

int len,i;

glRasterPos2f(x,y);

len=(int)strlen(s);

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

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18,s[i]);

}

void text2(int x,int y,char s)

{

glRasterPos2f(x,y);

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18,s);

}

void text1(int x,int y,char \*s)

{

int len,i;

glRasterPos2f(x,y);

len=(int)strlen(s);

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

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24,s[i]);

}

void text3(int x,int y,char s)

{

glRasterPos2f(x,y);

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24,s);

}

void starter()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,0.5,0.5);

text1(95,450,"DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT");

glColor3f(0.5,0.5,0.5);

text(270,425,"CG MINI PROJECT");

text(0,420,"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

glColor3f(0,0.7,0.7);

text1(275,370,"PROJECT ON");

glColor3f(0.5,0.7,0.5);

text1(285,345,"SIMULATION OF PROPELLERS");

text(0,325,"- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");

glColor3f(0.5,0.5,0.5);

text(250,260,"PROJECT ASSOCIATES");

text(0,255,"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

glColor3f(0.9,0.3,0.7);

text1(50,220,"SWAPNIL POWAR");

text1(430,220,"SWARAJ PAUL");

glColor3f(1.0,1.0,1.0);

text1(85,200,"1DT15CS421");

text1(442,200,"1DT14CS102");

glColor3f(1,1,0);

text(270,100,"Press SPACE to begin");

glFlush();

glutSwapBuffers();

}

//for cubical blades

void DrawTopCubeSet()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, white);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, white);

glBegin(GL\_POLYGON);

glEnd();

}

void DrawTopCubeSetNormalVector()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, white);

glEnd();

}

void DrawTopTriangleSet()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, yellow);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, yellow);

glBegin(GL\_TRIANGLE\_STRIP);

for (int i = 180; i >= 0; i = i - numberOfTriangles)

{

x = i\*PI / 180;

y = 0.5\*sin(i\*PI / 180);

z = -sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glNormal3f(x, sin(yTwisted), cos(zTwisted));

glVertex3f(x, 0, y);

glVertex3f(x, yTwisted, zTwisted);

}

glEnd();

}

void DrawTopTriangleSetNormalVector()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, white);

for (int i = 180; i >= 0; i = i - numberOfTriangles-5)

{

x = i\*PI / 180;

y = 0.5\*sin(i\*PI / 180);

z = -sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glLineWidth(2.5);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex3f(x, 0.0, 0.0);

glVertex3f(x, sin(yTwisted), cos(zTwisted));

glEnd();

}

}

void DrawBottomTriangleSet()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, yellow);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, yellow);

glBegin(GL\_TRIANGLE\_STRIP);

for (int i = 180; i >= 0; i = i - numberOfTriangles)

{

x = i\*PI / 180;

y = -0.5\*sin(i\*PI / 180);

z = sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glNormal3f(x, sin(yTwisted), cos(zTwisted));

glVertex3f(x, yTwisted, zTwisted);

glVertex3f(x, 0, -y);

}

glEnd();

}

void DrawBottomTriangleSetNormalVector()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, white);

for (int i = 180; i >= 0; i = i - numberOfTriangles-5)

{

x = i\*PI / 180;

y = -0.5\*sin(i\*PI / 180);

z = sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glLineWidth(2.5);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex3f(x, 0.0, 0.0);

glVertex3f(x, sin(yTwisted), cos(zTwisted));

glEnd();

}

}

void DrawBackTriangleSet()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, yellow);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, yellow);

glBegin(GL\_TRIANGLE\_STRIP);

for (int i = 180; i >= 0; i = i - numberOfTriangles)

{

x = i\*PI / 180;

y = 0.5\*sin(i\*PI / 180);

z = -sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glNormal3f(x, sin(yTwisted), -cos(zTwisted));

glVertex3f(x, yTwisted, zTwisted);

x = i\*PI / 180;

y = -0.5\*sin(i\*PI / 180);

z = sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glNormal3f(x, sin(yTwisted), -cos(zTwisted));

glVertex3f(x, yTwisted, zTwisted);

}

glEnd();

}

void DrawBackTriangleSetNormalVector()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, white);

for (int i = 180; i >= 0; i = i - numberOfTriangles-5)

{

x = i\*PI / 180;

y = 0.5\*sin(i\*PI / 180);

z = -sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glLineWidth(2.5);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex3f(x, 0.0, 0.0);

glVertex3f(x, sin(yTwisted), -cos(zTwisted));

glEnd();

x = i\*PI / 180;

y = -0.5\*sin(i\*PI / 180);

z = sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glLineWidth(2.5);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex3f(x, 0.0, 0.0);

glVertex3f(x, sin(yTwisted), -cos(zTwisted));

glEnd();

}

}

void DrawInsideTriangleSet()

{

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glBegin(GL\_TRIANGLE\_STRIP);

for (int i = 180; i >= 0; i = i - numberOfTriangles)

{

x = i\*PI / 180;

y = 0.5\*sin(i\*PI / 180);

z = -sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glVertex3f(x, yTwisted, zTwisted);

glVertex3f(x, 0, y);

x = i\*PI / 180;

y = -0.5\*sin(i\*PI / 180);

z = sin(i\*PI / 180);

twistAng = twistConstant \* x;

twistCos = cos(twistAng);

twistSin = sin(twistAng);

yTwisted = twistCos \* y - twistSin \* z;

zTwisted = twistSin \* y + twistCos \* z;

glVertex3f(x, yTwisted, zTwisted);

}

glEnd();

}

void writemessage()

{

printf(" X => x++ <= Move light source in direction of +X\n");

printf(" Y => y++ <= Move light source in direction of +Y\n");

printf(" Z => z++ <= Move light source in direction of +Z\n");

printf("\n");

printf("^X => x-- <= Move light source in direction of -X\n");

printf("^Y => y-- <= Move light source in direction of -Y\n");

printf("^Z => z-- <= Move light source in direction of -Z\n");

printf("\n");

printf(" ^ => Move camera up\n");

printf(" > => Move camera right\n");

printf(" < => Move camera left\n");

printf(" down arrow => Move camera down\n");

printf("\n");

printf(" t => More Twist\n");

printf(" f => Less Twist\n");

printf("\n");

printf(" q => More Propeller\n");

printf(" a => Less Propeller\n");

printf("\n");

printf(" w => More Triangles\n");

printf(" s => Less Triangles\n");

printf("\n");

printf(" 0 => Toggling light source\n");

printf("\n");

printf(" r => Rotates Propeller\n");

printf("\n");

printf(" n => Toggle to show Normal Vectors\n");

printf("\n");

printf(" You can not move the light source when the light source is off !!!\n");

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (GLfloat)w / (GLfloat)h, 0.01, 20.0);

glMatrixMode(GL\_MODELVIEW);

}

void display(void)

{

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

glLoadIdentity();

cpos[0] = zoom \* cos(beta) \* sin(alpha);

cpos[1] = zoom \* sin(beta);

cpos[2] = zoom \* cos(beta) \* cos(alpha);

gluLookAt(cpos[0], cpos[1], cpos[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

if (lightSource == true)

{

glLightfv(GL\_LIGHT0, GL\_POSITION, lpos);

glMaterialfv(GL\_FRONT, GL\_EMISSION, white);

glPushMatrix();

glTranslatef(lpos[0], lpos[1], lpos[2]);

glutSolidSphere(0.1, 10, 8);

glPopMatrix();

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

}

glPushMatrix();

glTranslated(-3,0,0);

glRotatef(rotateConstant, 0, 0, 1);

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

{

glPushMatrix();

glRotatef(i \* 360 / numberOfObj, 0, 0, 1);

DrawTopTriangleSet();

DrawBottomTriangleSet();

DrawBackTriangleSet();

DrawInsideTriangleSet();

if (showNormalVectors)

{

DrawTopTriangleSetNormalVector();

DrawBottomTriangleSetNormalVector();

DrawBackTriangleSetNormalVector();

DrawTopTriangleSetNormalVector();

}

glPopMatrix();

}

// Cone

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, red);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, red);

glPushMatrix();

glTranslated(0, 0, -1.5);

glutSolidCone(1, 2, 50, 50);

glPopMatrix();

// Back of Cone

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, red);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, red);

glBegin(GL\_POLYGON);

glNormal3f(0, 0, 1);

for (int i = 0; i <= 360; i++)

{

glVertex3f(cos(i\*PI / 180) \* 1, sin(i\*PI / 180) \* 1, -1.5);

}

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(3,0,0);

glRotatef(rotateConstant, 0, 0, 1);

DrawTopCubeSet();

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

{

glPushMatrix();

glRotatef(i \* 360 / numberOfObj, 0, 0,1 );

glScaled(.2,2,4);

glutSolidCube(.8);

glRotatef(i \* 360 / numberOfObj, 0, 0, 1);

glPopMatrix();

}

// Cone

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, red);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, red);

glPushMatrix();

glTranslated(0, 0, -1.5);

glutSolidCone(1, 2, 50, 50);

glPopMatrix();

// Back of Cone

glMaterialfv(GL\_FRONT, GL\_EMISSION, black);

glMaterialfv(GL\_BACK, GL\_EMISSION, black);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, red);

glMaterialfv(GL\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, red);

glBegin(GL\_POLYGON);

glNormal3f(0, 0, 1);

for (int i = 0; i <= 360; i++)

{

glVertex3f(cos(i\*PI / 180) \* 1, sin(i\*PI / 180) \* 1, -1.5);

}

glEnd();

glPopMatrix();

glutSwapBuffers();

glFlush();

}

void keyboard(unsigned char key, int x, int y)

{

int i;

if(key==' ')

glutDisplayFunc(display);

switch (key)

{

case 27:

exit(0);

break;

case 'x':

if (lightSource == true)

lpos[0] = lpos[0] + 0.2;

glutPostRedisplay();

break;

case 'X':

if (lightSource == true)

lpos[0] = lpos[0] - 0.2;

glutPostRedisplay();

break;

case 'y':

if (lightSource == true)

lpos[1] = lpos[1] + 0.2;

glutPostRedisplay();

break;

case 'Y':

if (lightSource == true)

lpos[1] = lpos[1] - 0.2;

glutPostRedisplay();

break;

case 'z':

if (lightSource == true)

lpos[2] = lpos[2] + 0.2;

glutPostRedisplay();

break;

case 'Z':

if (lightSource == true)

lpos[2] = lpos[2] - 0.2;

glutPostRedisplay();

break;

case '+':

if (zoom != 1.5)zoom = zoom - 0.5;

glutPostRedisplay();

break;

case '-':

if (zoom != 15)zoom = zoom + 0.5;

glutPostRedisplay();

break;

case '0':

if (lightSource == true){

glDisable(GL\_LIGHT0);

lightSource = false;

}

else{

glEnable(GL\_LIGHT0);

lightSource = true;

}

glutPostRedisplay();

break;

case 't':

if (twistConstant <= PI / 4){

twistConstant = twistConstant + 0.05;

glutPostRedisplay();

}

break;

case 'f':

if (twistConstant >= 0){

twistConstant = twistConstant - 0.05;

glutPostRedisplay();

}

break;

case 'r':

for(i=0;i<999;i++)

{

rotateConstant = rotateConstant + 5;

glutPostRedisplay();

}

break;

case 'q':

if (numberOfObj <= 6){

numberOfObj++;

glutPostRedisplay();

}

break;

case 'a':

if (numberOfObj >= 0){

numberOfObj--;

glutPostRedisplay();

}

break;

case 's':

if (numberOfTriangles < 90){

numberOfTriangles++;

glutPostRedisplay();

}

break;

case 'w':

if (numberOfTriangles > 1){

numberOfTriangles--;

glutPostRedisplay();

}

case 'n':

if (showNormalVectors)

showNormalVectors = false;

else

showNormalVectors = true;

glutPostRedisplay();

break;

default:

break;

}

}

void specialkey(GLint key, int x, int y)

{

switch (key) {

case GLUT\_KEY\_RIGHT:

alpha = alpha + PI / 180;

if (alpha > 2 \* PI) alpha = alpha - 2 \* PI;

glutPostRedisplay();

break;

case GLUT\_KEY\_LEFT:

alpha = alpha - PI / 180;

if (alpha < 0) alpha = alpha + 2 \* PI;

glutPostRedisplay();

break;

case GLUT\_KEY\_UP:

if (beta < 0.45\*PI) beta = beta + PI / 180;

glutPostRedisplay();

break;

case GLUT\_KEY\_DOWN:

if (beta > -0.05\*PI) beta = beta - PI / 180;

glutPostRedisplay();

break;

default:

break;

}

}

void mouse(int button,int state,int x,int y)

{

if(button==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN)

exit(0);

if(button==GLUT\_MIDDLE\_BUTTON && state==GLUT\_DOWN)

exit(0);

if(button==GLUT\_RIGHT\_BUTTON && state==GLUT\_DOWN)

exit(0);

}

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

{

writemessage();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(1200, 800);

glutInitWindowPosition(0, 0);

glutCreateWindow(argv[0]);

glClearColor(0.137255,0.419608,0.556863, 0.0);

glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_LIGHTING);

glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

glEnable(GL\_LIGHT0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0.0, 5.0, 10.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutSpecialFunc(specialkey);

glutMouseFunc(mouse);

glutMainLoop();

return 0;

}