# Đua xe

#include <iostream>

#include <stdlib.h>

#include <math.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include<iostream>

#include<sstream>

#include <string>

#include<string.h>

#include<windows.h>

#include<mmsystem.h>

using namespace std;

static float xp=-.55;

int static crspeed = 60;

float crmove = 4;

float static y11= -3.3;

float static zp= 2;

float static tpx= .15;

int static carpos= 0;

float view = 10.0;

int static score= 0;

int static totalMeter = 0;

char quote[6][80];

int numberOfQuotes = 0, i;

int static carspeed= 45;

float static sky\_red=0;

float static sky\_green= .8;

float sky\_blue= 1.0;

int roadlight = 50;

void sprint( float x, float y, string st)

{

int l,i;

// l=strlen( st ); // see how many characters are in text string.

glColor3f(0.0,0.0,0.0);

//glDisable(GL\_LIGHTING);

glRasterPos2f( x, y); // location to start printing text

for( i=0; i < st.length(); i++) // loop until i is greater then l

{

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

}

}

void keyboardown(int key, int x, int y)

{

switch (key){

case GLUT\_KEY\_RIGHT:

xp=.55;

carpos=1;

break;

case GLUT\_KEY\_LEFT:

xp=-.55;

carpos=0;

break;

case GLUT\_KEY\_UP:

if (crspeed>5){

crspeed-=5;

carspeed+=5;

}

else

crspeed=crspeed;

break;

case GLUT\_KEY\_DOWN:

if (crspeed<60){

crspeed+=5;

carspeed-=5;

}

else

crspeed=crspeed;

break;

default:

break;

}

}

//Extra Display Function

void reshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60, 1.0, 1.0, 3200);

glMatrixMode(GL\_MODELVIEW);

}

GLfloat UpwardsScrollVelocity = -1.0;

void timeTick(void)

{

if (UpwardsScrollVelocity< -1)

view -= 0.0011;

if (view < 0) { view = 2; UpwardsScrollVelocity = -1.0; }

// exit(0);

UpwardsScrollVelocity -= 0.2;

glutPostRedisplay();

}

void RenderToDisplay()

{

int l, lenghOfQuote, i;

glTranslatef(0.0, 0.0, 0.0);

glRotatef(-20, 1.0, 0.0, 0.0);

glScalef(0.05, 0.05, 0.05);

for (l = 0; l<numberOfQuotes; l++)

{

lenghOfQuote = (int)strlen(quote[l]);

glPushMatrix();

glTranslatef(-(lenghOfQuote \* 37), (l \* 200), 0.0);

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

{

glColor3f((UpwardsScrollVelocity / 10) + 300 + (l \* 10), (UpwardsScrollVelocity / 10) + 300 + (l \* 10), 0.0);

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, quote[l][i]);

}

glPopMatrix();

}

}

void myDisplayFunction(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

gluLookAt(0.0, 30.0, 100.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

RenderToDisplay();

glutSwapBuffers();

}

int winner(char a)

{

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

glutInitWindowSize(800, 500);

glutCreateWindow("GAME OVER");

glClearColor(0.0, 0.0, 0.0, 1.0);

glLineWidth(3);

//char c=char(score);

strcpy(quote[1], "Game Over");

strcpy(quote[0], "Nirapod Sorok Chai");

numberOfQuotes = 5;

glutDisplayFunc(myDisplayFunction);

glutReshapeFunc(reshape);

glutIdleFunc(timeTick);

glutMainLoop();

return 0;

}

//Extra Display Function End

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

{

if(key=='n')

{

sky\_red=0;

sky\_green=0.2;

sky\_blue=0.25;

roadlight= 255;

}

else if(key=='d')

{

sky\_red=0;

sky\_green=0.8;

sky\_blue=1.0;

roadlight= 50;

}

else if(key=='e')

{

exit(1);

}

}

bool collision()

{

if (0 > (crmove-y11)&& -1 < (crmove-y11)&& carpos==1 )

{

return true;

}

else if (0 > (crmove-(y11-10))&& -1 < (crmove-(y11-10))&& carpos==0 )

{

return true;

}

else if (0 > (crmove-(y11-20))&& -1 < (crmove-(y11-20))&& carpos==0 )

{

return true;

}

else if (0 > (crmove-(y11-30))&& -1 < (crmove-(y11-30))&& carpos==1 )

{

return true;

}

else if (0 > (crmove-(y11-30))&& -1 < (crmove-(y11-30))&& carpos==0 )

{

y11-=40;

return false;

}

else

{

return false;

}

}

bool GameScore()

{

if (0 > (crmove-y11)&& -1 < (crmove-y11)&& carpos==0 )

{

score++;

}

else if (0 > (crmove-(y11-10))&& -1 < (crmove-(y11-10))&& carpos==1 )

{

score++;

}

else if (0 > (crmove-(y11-20))&& -1 < (crmove-(y11-20))&& carpos==1 )

{

score++;

}

else if (0 > (crmove-(y11-30))&& -1 < (crmove-(y11-30))&& carpos==0 )

{

//return false;

score+=10;

}

else if (0 > (crmove-(y11-35))&& -1 < (crmove-(y11-35))&& carpos==0 )

{

return true;

score+=10;

}

else{

totalMeter++;

// cout<<totalMeter<<"Total Meter"<<endl;

return false;

}

}

//Initializes 3D rendering

void initRendering() {

glEnable(GL\_DEPTH\_TEST);

}

//Called when the window is resized

void handleResize(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);

}

float rtri =0;

float \_angle = 0.0;

float \_cameraAngle = 0.0;

float \_ang\_tri = 0.0;

float home= 6;

void gamercar()

{

glPushMatrix();

glTranslatef(xp, -1.0, 3.5);

glPushMatrix();

glRotatef(-10 , -1.0, 0.0, 0.0);

glScalef(1.3, 0.8, 2.0);

glColor3f(1, 1, 1);

glutSolidCube(.3);

glPopMatrix();

glPushMatrix();

glTranslatef(0.0, tpx, 0.0);

glRotatef(-10 , -1.0, 0.0, 0.0);

glScalef(1.0, .5, .5);

glColor3f(0, 0, 0);

glutSolidCube(.2);

glPopMatrix();

glPushMatrix();

glTranslatef(0.21, -0.12, 0.10);

glRotatef(80 , 0.0, 1.2, 0.0);

glScalef(.07, .14, .07);

glColor3f(1, 0, 0);

glutWireTorus(.2, .4, 20, 15);

glPopMatrix();

glPushMatrix();

glTranslatef(0.21, 0.0, -0.2);

glRotatef(80 , 0.0, 1.2, 0.0);

glScalef(.07, .14, .07);

glColor3f(1, 0, 0);

glutWireTorus(.2, .4, 20, 15);

glPopMatrix();

glPushMatrix();

glTranslatef(-0.21, -0.12, 0.10);

glRotatef(80 , 0.0, 1.2, 0.0);

glScalef(.07, .14, .07);

glColor3f(1, 0, 0);

glutWireTorus(.2, .4, 20, 15);

glPopMatrix();

glPushMatrix();

glTranslatef(-0.21, 0.0, -0.2);

glRotatef(80 , 0.0, 1.2, 0.0);

glScalef(.07, .14, .07);

glColor3f(1, 0, 0);

glutWireTorus(.2, .4, 20, 15);

glPopMatrix();

glPushMatrix();

glTranslatef(-0.10, -0.10, 0.30);

glRotatef(80 , 0.0, 1.2, 0.0);

glScalef(.07, .03, .07);

glColor3f(1, .5, 0);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslatef(0.10, -0.10, 0.30);

glRotatef(80 , 0.0, 1.2, 0.0);

glScalef(.07, .03, .07);

glColor3f(1, .5, 0);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

}

void sky()

{

glBegin(GL\_QUADS);

glColor3ub(sky\_red, 0, sky\_blue);

glVertex3f(-4.0, 1.5, 0);

glVertex3f(4.0, 1.5, 0);

glVertex3f(8.0, 3, 0);

glVertex3f(-8.0, 3, 0);

glEnd();

}

void roadside()

{

//tree

for (float z = -38; z < 400; z +=4)

{

glPushMatrix();

glColor3ub(200, 200, 200);

glTranslatef(-1.20, z, .40);

glScalef(.2, .2, 3);

glutSolidCube(.4);

glPopMatrix();

glPushMatrix();

glColor3ub(200, 200, 200);

glTranslatef(-.84, z, 1.0);

glScalef(2, .2, .2);

glutSolidCube(.4);

glPopMatrix();

glPushMatrix();

glColor4ub(255, 255, 255, roadlight);

glTranslatef(-.44, z, 0.70);

//glScalef(2, .2, .2);

glutSolidCone(.2,.3,15,20);

glPopMatrix();

//2d red Bar

/\*

glPushMatrix();

glColor3ub(255, 0, 20);

glTranslatef(-1.4, z+.2, .15);

glScalef(.80, .30, .60);

glutSolidCube(.8);

glPopMatrix();

glPushMatrix();

glColor3ub(255, 255, 20);

glTranslatef(0.0, 0.0, .40);

glBegin(GL\_QUADS);

glVertex3f(-1.10, z, 0);

glVertex3f(-.85, z, -.2);

glVertex3f(-.85, z+.3, -.2);

glVertex3f(-1.10, z+.3, 0);

glEnd();

glPopMatrix();

glPushMatrix();

glColor3ub(255, 0, 20);

glTranslatef(-.7, z+.2, .10);

glScalef(1.0, .60, .60);

glutSolidCube(.4);

glPopMatrix();

\*/

}

glBegin(GL\_QUADS);

glColor3ub(0, 155, 20);

glVertex3f(-5.0, -10, 0);

glVertex3f(-1.0, -10, 0);

glVertex3f(-1.0, 400, 0);

glVertex3f(-5.0, 400, 0);

glEnd();

glBegin(GL\_QUADS);

glColor3ub(0, 155, 20);

glVertex3f(1.0, -10, 0);

glVertex3f(5.0, -10, 0);

glVertex3f(5.0, 400, 0);

glVertex3f(1.0, 400, 0);

glEnd();

}

void house(){

for (float z = -40; z < 400; z +=4.8)

{

glPushMatrix();

glScalef(1.0, 1.0, 1.0);

glPushMatrix();

glColor3ub(70, 61, 46);

glTranslatef(-3.0, z, .30);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glColor3ub(255, 255, 255);

glTranslatef(-2.55, z, .40);

glutSolidCube(.2);

glPopMatrix();

glPushMatrix();

glColor3ub(255, 255, 255);

glTranslatef(-2.9, z-.5, .2);

glScalef(.6, .2, 1);

glutSolidCube(.5);

glPopMatrix();

glPushMatrix();

glColor3f(0, 1, 1);

glRotatef(0, 0.0, 0.0, 0.0);

glTranslatef(-3.0, z, .70);

glutSolidCone(1, 1, 4, 6);

glPopMatrix();

glPopMatrix();

//right side

glPushMatrix();

glPushMatrix();

glColor3f(1, 1, 1);

glTranslatef(3.0, z, .30);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glColor3ub(0, 0, 0);

glTranslatef(2.55, z, .40);

glutSolidCube(.2);

glPopMatrix();

glPushMatrix();

glColor3ub(0, 0, 0);

glTranslatef(2.9, z-.5, .2);

glScalef(.6, .2, 1);

glutSolidCube(.5);

glPopMatrix();

glPushMatrix();

glColor3ub(70, 61, 46);

glTranslatef(3.0, z, .60);

glutSolidCone(1, .8, 4, 6);

glPopMatrix();

glPopMatrix();

}

}

void tree(){

for (float z = -40; z < 400; z +=4)

{

glPushMatrix();

glPushMatrix();

glColor3f(0, 1, 0);

glTranslatef(-1.20, z, .45);

glutSolidCone(.2, .4, 20, 10);

glPopMatrix();

glColor3ub(102, 51, 0);

glTranslatef(-1.20, z, .25);

glScalef(.2, .2, 1);

glutSolidCube(.4);

glPopMatrix();

//right

glPushMatrix();

glPushMatrix();

glColor3f(0, 1, 0);

glTranslatef(1.20, z, .50);

glutSolidCone(.2, .4, 20, 10);

glPopMatrix();

glColor3ub(102, 51, 0);

glTranslatef(1.20, z, .30);

glScalef(.2, .2, 1);

glutSolidCube(.4);

glPopMatrix();

}

}

void roadline(){

for (float z = -4; z < 6; z +=1)

{

glPushMatrix();

glColor3f(1, 1, 1);

glBegin(GL\_QUADS);

glVertex3f(-.03, z, 0);

glVertex3f(.03, z, 0);

glVertex3f(.03, z+.5, 0);

glVertex3f(-.03, z+.5, 0);

glEnd();

glPopMatrix();

}

}

void objectcube()

{

for (float zp = -20; zp < 400; zp+=40)

{

//glPushMatrix();

glPushMatrix();

glColor3f(0, 1, 0);

glTranslatef(-.50, zp, -.1);

glutSolidCube(.4);

glPopMatrix();

glPushMatrix();

glColor3f(0, 1, 0);

glTranslatef(.50, zp+10, -.1);

glutSolidCube(.4);

glPopMatrix();

glPushMatrix();

glColor3f(0, 1, 0);

glTranslatef(.50, zp+20, -.1);

glutSolidCube(.4);

glPopMatrix();

glPushMatrix();

glColor3f(0, 1, 0);

glTranslatef(-.50, zp+30, -.1);

glutSolidCube(.4);

glPopMatrix();

}

}

void road(){

for (float z = -10; z < 400; z +=1)

{

glPushMatrix();

glColor3f(1, 1, 1);

glBegin(GL\_QUADS);

glVertex3f(-.03, z, 0);

glVertex3f(.03, z, 0);

glVertex3f(.03, z+.5, 0);

glVertex3f(-.03, z+.5, 0);

glEnd();

glPopMatrix();

}

glPushMatrix();

glColor3ub(0, 0, 0);

glTranslatef(0.0, 0.0, -.50);

glBegin(GL\_QUADS);

glVertex3f(-1.3, -10, 0);

glVertex3f(1.3, -10, 0);

glVertex3f(1.3, 400, 0);

glVertex3f(-1.3, 400, 0);

glEnd();

glPopMatrix();

}

//Draws the 3D scene

void drawScene() {

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

glMatrixMode(GL\_MODELVIEW); //Switch to the drawing perspective

glLoadIdentity(); //Reset the drawing perspective

glRotatef(-\_cameraAngle, 0.0, 1.0, 0.0); //Rotate the camera

glTranslatef(0.0, 0.0, -7.0); //Move forward 5 units

gamercar();

//sky();

glPushMatrix(); //Save the current state of transformations

glTranslatef(0.0, 0.0, 0.0); //Move to the center of the triangle

glRotatef(80 , -1.0, 0.0, 0.0);

glPushMatrix();

glTranslatef(0.0, crmove, 0.0);

glClearColor(0.0, 0.0, 0.0, 1.0);

road();

glPopMatrix();

glPushMatrix();

glTranslatef(0.0, crmove, 0.0);

tree();

house();

roadside();

objectcube();

GameScore();

cout<<score<<endl;

//cout<<crmove<<endl;

//cout<<y11<<endl;

glPopMatrix();

glPushMatrix();

glColor3ub(0,0,0);

glTranslatef(5.52, 0.0, 2.0);

ostringstream cnvrt;

cnvrt << score;

sprint(-4,-2.3,"Score: "+cnvrt.str());

glPopMatrix();

glPushMatrix();

glColor3ub(0,0,0);

glTranslatef(5.5, 0.0, 1.8);

ostringstream cnvrt2;

cnvrt2 << totalMeter;

sprint(-4,-2.4,"Distance Travel: "+cnvrt2.str());

glPopMatrix();

glPushMatrix();

glColor3ub(0,0,0);

glTranslatef(5.5, 0.0, 1.6);

ostringstream cnvrt3;

cnvrt3 << carspeed;

sprint(-4,-2.4,"Speed: "+cnvrt3.str());

glPopMatrix();

glPopMatrix();

glClearColor(sky\_red, sky\_green, sky\_blue, 1.0);

if(collision())

{

winner('a');

}

//sky();

glutSwapBuffers();

}

void update(int value) {

crmove-= 0.1f;

\_angle += 2.0f;

if (\_angle > 360) {

\_angle -= 360;

}

\_ang\_tri += 0.7f;

if (\_ang\_tri > 80) {

\_ang\_tri=0;

//crmove= 40;

}

glutPostRedisplay(); //Tell GLUT that the display has changed

//Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(crspeed, update, 0);

}

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

//Initialize GLUT

glutInit(&argc, argv);

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

glutInitWindowSize(800, 500);

glutInitWindowPosition(100,100);

glutCreateWindow("Transformations");

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glEnable( GL\_BLEND );

//glClearColor(1.0, 1.0, 1.0, 0.0);

initRendering();

//Set handler functions

glutDisplayFunc(drawScene);

glutReshapeFunc(handleResize);

glutTimerFunc(25, update, 0); //Add a timer

glutKeyboardFunc(keyboard);

glutSpecialFunc(keyboardown);

glutMainLoop();

return 0;

}

# Bắn súng không gian

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* COMPUTER GRAPHICS PROJECT -> THE SPACESHIP SHOOTING GAME

\*\*

\*\* BY

\*\*

\*\* Kaushik Jeyaraman & Nikilesh Iyer

\*\*

\*\* for any queries

\*\* e-mail me : kaushikjjj@gmail.com

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* G A M E D E S C R I P T I O N S

\*\*

\*\* keybord controls= w,a,s,d

\*\* mouse left click to fire

\*\*

\*\* INSTRUCTIONS:

\*\* Dodge the objects and shoot them down

\*\*

\*\* OBJECTIVE : Beat the high score.

\*\* Score +1 point for shooting each object

\*\* Score +50 for lvl up

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <gl/glut.h>

#include <math.h>

#include<string.h>

#define PI 3.14159

#define GAME\_SCREEN 0 //Constant to identify background color

#define MENU\_SCREEN 4

#define MAX\_STONES 100

#define MAX\_STONE\_TYPES 5

#define stoneRotationSpeed 20

#define SPACESHIP\_SPEED 20

int stoneTranslationSpeed=5;

GLint m\_viewport[4];

GLint CI=0;

int x,y;

int i;

int randomStoneIndices[100];

int index;

int Score=0;

int alienLife=100;

int GameLvl= 1;

float mouseX ,mouseY ; //Cursor coordinates;

float LaserAngle=0 ,stoneAngle =0,lineWidth = 1;

float xOne=0,yOne=0; //Spaceship coordinates

float xStone[MAX\_STONES] ,yStone[MAX\_STONES];//coordinates of stones

float xHealthBarStart = 1200; //Health bar starting coodinate

GLint stoneAlive[MAX\_STONES]; //check to see if stone is killed

bool mButtonPressed= false,startGame=false,gameOver=false; //boolean values to check state of the game

bool startScreen = true ,nextScreen=false,previousScreen=false;

bool gameQuit = false,instructionsGame = false, optionsGame = false;

GLfloat a[][2]={0,-50, 70,-50, 70,70, -70,70};

GLfloat LightColor[][3]={1,1,0, 0,1,1, 0,1,0};

GLfloat AlienBody[][2]={{-4,9}, {-6,0}, {0,0}, {0.5,9}, {0.15,12}, {-14,18}, {-19,10}, {-20,0},{-6,0}};

GLfloat AlienCollar[][2]={{-9,10.5}, {-6,11}, {-5,12}, {6,18}, {10,20}, {13,23}, {16,30}, {19,39}, {16,38},

{10,37}, {-13,39}, {-18,41}, {-20,43}, {-20.5,42}, {-21,30}, {-19.5,23}, {-19,20},

{-14,16}, {-15,17},{-13,13}, {-9,10.5}};

GLfloat ALienFace[][2]={{-6,11}, {-4.5,18}, {0.5,20}, {0.,20.5}, {0.1,19.5}, {1.8,19}, {5,20}, {7,23}, {9,29},

{6,29.5}, {5,28}, {7,30}, {10,38},{11,38}, {11,40}, {11.5,48}, {10,50.5},{8.5,51}, {6,52},

{1,51}, {-3,50},{-1,51}, {-3,52}, {-5,52.5}, {-6,52}, {-9,51}, {-10.5,50}, {-12,49}, {-12.5,47},

{-12,43}, {-13,40}, {-12,38.5}, {-13.5,33},{-15,38},{-14.5,32}, {-14,28}, {-13.5,33}, {-14,28},

{-13.8,24}, {-13,20}, {-11,19}, {-10.5,12}, {-6,11} } ;

GLfloat ALienBeak[][2]={{-6,21.5}, {-6.5,22}, {-9,21}, {-11,20.5}, {-20,20}, {-14,23}, {-9.5,28}, {-7,27}, {-6,26.5},

{-4.5,23}, {-4,21}, {-6,19.5}, {-8.5,19}, {-10,19.5}, {-11,20.5} };

char highScore[100],ch;

void display();

void StoneGenerate();

void displayRasterText(float x ,float y ,float z ,char \*stringToDisplay) {

int length;

glRasterPos3f(x, y, z);

length = strlen(stringToDisplay);

for(int i=0 ;i<length ;i++){

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

}

}

void SetDisplayMode(int modeToDisplay) {

switch(modeToDisplay){

case GAME\_SCREEN: glClearColor(0, 0, 0, 1);break;

case MENU\_SCREEN : glClearColor(1, 0 , 0, 1);break;

}

}

void initializeStoneArray() {

//random stones index

for(int i = 0;i < MAX\_STONES ;i++) {

randomStoneIndices[i]=rand()%MAX\_STONE\_TYPES;

stoneAlive[i]=true;

}

xStone[0] = -(200\*MAX\_STONES)-600; //START LINE for stone appearance

for(int i = 0;i<MAX\_STONES ;i++) { //ramdom appearance yIndex for each stone

yStone[i]=rand()%600;

if(int(yStone[i])%2)

yStone[i]\*=-1;

xStone[i+1] = xStone[i] + 200; //xIndex of stone aligned with 200 units gap

}

}

void DrawAlienBody()

{

glColor3f(0,1,0); //BODY color

glBegin(GL\_POLYGON);

for(i=0;i<=8;i++)

glVertex2fv(AlienBody[i]);

glEnd();

glColor3f(0,0,0); //BODY Outline

glLineWidth(1);

glBegin(GL\_LINE\_STRIP);

for(i=0;i<=8;i++)

glVertex2fv(AlienBody[i]);

glEnd();

glBegin(GL\_LINES); //BODY effect

glVertex2f(-13,11);

glVertex2f(-15,9);

glEnd();

}

void DrawAlienCollar()

{

glColor3f(1,0,0); //COLLAR

glBegin(GL\_POLYGON);

for(i=0;i<=20 ;i++)

glVertex2fv(AlienCollar[i]);

glEnd();

glColor3f(0,0,0); //COLLAR outline

glBegin(GL\_LINE\_STRIP);

for(i=0;i<=20 ;i++)

glVertex2fv(AlienCollar[i]);

glEnd();

}

void DrawAlienFace()

{

//glColor3f(0.6,0.0,0.286); //FACE

//glColor3f(0.8,0.2,0.1);

//glColor3f(0,0.5,1);

glColor3f(0,0,1);

glBegin(GL\_POLYGON);

for(i=0;i<=42 ;i++)

glVertex2fv(ALienFace[i]);

glEnd();

glColor3f(0,0,0); //FACE outline

glBegin(GL\_LINE\_STRIP);

for(i=0;i<=42 ;i++)

glVertex2fv(ALienFace[i]);

glEnd();

glBegin(GL\_LINE\_STRIP); //EAR effect

glVertex2f(3.3,22);

glVertex2f(4.4,23.5);

glVertex2f(6.3,26);

glEnd();

}

void DrawAlienBeak()

{

glColor3f(1,1,0); //BEAK color

glBegin(GL\_POLYGON);

for(i=0;i<=14 ;i++)

glVertex2fv(ALienBeak[i]);

glEnd();

glColor3f(0,0,0); //BEAK outline

glBegin(GL\_LINE\_STRIP);

for(i=0;i<=14 ;i++)

glVertex2fv(ALienBeak[i]);

glEnd();

}

void DrawAlienEyes()

{

glColor3f(0,1,1);

glPushMatrix();

glRotated(-10,0,0,1);

glTranslated(-6,32.5,0); //Left eye

glScalef(2.5,4,0);

glutSolidSphere(1,20,30);

glPopMatrix();

glPushMatrix();

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

glTranslated(-8,36,0); //Right eye

glScalef(2.5,4,0);

glutSolidSphere(1,100,100);

glPopMatrix();

}

void DrawAlien()

{

DrawAlienBody();

DrawAlienCollar();

DrawAlienFace();

DrawAlienBeak();

DrawAlienEyes();

}

void DrawSpaceshipBody()

{

glColor3f(1,0,0); //BASE

glPushMatrix();

glScalef(70,20,1);

glutSolidSphere(1,50,50);

glPopMatrix();

glPushMatrix(); //LIGHTS

glScalef(3,3,1);

glTranslated(-20,0,0); //1

glColor3fv(LightColor[(CI+0)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //2

glColor3fv(LightColor[(CI+1)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //3

glColor3fv(LightColor[(CI+2)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //4

glColor3fv(LightColor[(CI+0)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //5

glColor3fv(LightColor[(CI+1)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //6

glColor3fv(LightColor[(CI+2)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //7

glColor3fv(LightColor[(CI+0)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //8

glColor3fv(LightColor[(CI+1)%3]);

glutSolidSphere(1,1000,1000);

glTranslated(5,0,0); //9

glColor3fv(LightColor[(CI+2)%3]);

glutSolidSphere(1,1000,1000);

glPopMatrix();

}

void DrawSteeringWheel()

{

glPushMatrix();

glLineWidth(3);

glColor3f(0.20,0.,0.20);

glScalef(7,4,1);

glTranslated(-1.9,5.5,0);

glutWireSphere(1,8,8);

glPopMatrix();

}

void DrawSpaceshipDoom()

{

glColor4f(0.7,1,1,0.0011);

glPushMatrix();

glTranslated(0,30,0);

glScalef(35,50,1);

glutSolidSphere(1,50,50);

glPopMatrix();

}

void DrawSpaceShipLazer() {

glColor3f(1, 0, 0);

glPushMatrix();

glBegin(GL\_POLYGON); //Lazer stem

glVertex2f(-55 ,10);

glVertex2f(-55 ,30);

glVertex2f(-50, 30);

glVertex2f(-50 ,10);

glEnd();

float xMid =0,yMid =0;

//Mid point of the lazer horizontal

xMid = (55+50)/2.0;

yMid = (25+35)/2.0;

//Rotating about the point ,20

glTranslated(-xMid, yMid, 0);

glRotated(LaserAngle, 0, 0 ,1);

glTranslated(xMid , -yMid ,0);

//find mid point of top of lazer stem

float midPoint = -(55+50)/2.0;

glBegin(GL\_POLYGON); //Lazer horizontal stem

glVertex2f(midPoint + 10 ,25);

glVertex2f(midPoint + 10 ,35);

glVertex2f(midPoint - 10 ,35);

glVertex2f(midPoint - 10 ,25);

glEnd();

glPopMatrix();

}

void DrawLazerBeam() {

float xMid = -(55+50)/2.0;

float yMid = (25+35)/2.0;

float mouseXEnd = -((- mouseX) + xOne);

float mouseYEnd = -((- mouseY) + yOne);

glLineWidth(5); //----Laser beam width

glColor3f(1, 0, 0);

glBegin(GL\_LINES);

glVertex2f(xMid ,yMid);

glVertex2f(mouseXEnd ,mouseYEnd);

glEnd();

glLineWidth(1);

}

void DrawStone(int StoneIndex)

{

glPushMatrix();

glLoadIdentity();

switch(StoneIndex) //CHANGE INDEX VALUE FOR DIFFERENT STONE VARIETY;

{

case 0:

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glColor3f(0.4f, 0.0f, 0.0f);

glScalef(35,35,1);

glutSolidSphere(1,9,50);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(60,10,1);

glutSolidSphere(1,5,50);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(10,60,1);

glutSolidSphere(1,5,50);

break;

case 1:

glColor3f(1.0f, 0.8f, 0.8f);

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(15,20,1);

glutSolidSphere(1,9,50);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(40,5,1);

glutSolidSphere(1,5,50);

break;

case 2:

glColor3f(0.2f, 0.2f, 0.0f);

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(60,25,1);

glutSolidSphere(1,9,50);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(25,60,1);

glutSolidSphere(1,9,50);

break;

case 3:

glColor3f(0.8f, 0.8f, 0.1f);

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(35,10,1);

glutSolidSphere(1,10,7);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(50,20,1);

glutSolidSphere(1,5,50);

break;

case 4:

glColor3f(0.26f, 0.26f, 0.26f);

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(10,55,1);

glutSolidSphere(1,9,50);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(20,10,1);

glutSolidSphere(1,9,50);

glLoadIdentity();

glTranslated(xStone[index] , yStone[index] ,0);

glRotatef(stoneAngle+45 ,0, 0, 1);

glTranslated(0, 0, 0);

glScalef(25,10,1);

glutSolidSphere(1,9,50);

break;

}

glPopMatrix();

}

bool checkIfSpaceShipIsSafe() {

for(int i =0 ;i<MAX\_STONES ;i++) {

if(stoneAlive[i]&((xOne >= (xStone[i]/2 -70) && xOne <= (xStone[i]/2 + 70) && yOne >= (yStone[i]/2 -18 ) && yOne <= (yStone[i]/2 + 53)) || (yOne <= (yStone[i]/2 - 20) && yOne >= (yStone[i]/2 - 90) && xOne >= (xStone[i]/2 - 40) && xOne <= (xStone[i]/2 + 40))))

{

stoneAlive[i]=0;

return false;

}

}

return true;

}

void SpaceshipCreate(){

glPushMatrix();

glTranslated(xOne,yOne,0);

if(!checkIfSpaceShipIsSafe() && alienLife ){

alienLife-=10;

xHealthBarStart -= 230;

}

DrawSpaceshipDoom();

glPushMatrix();

glTranslated(4,19,0);

DrawAlien();

glPopMatrix();

DrawSteeringWheel();

DrawSpaceshipBody();

DrawSpaceShipLazer();

if(mButtonPressed) {

DrawLazerBeam();

}

glEnd();

glPopMatrix();

}

void DisplayHealthBar() {

glColor3f(1 ,0 ,0);

glBegin(GL\_POLYGON);

glVertex2f(-xHealthBarStart ,700);

glVertex2f(1200 ,700);

glVertex2f(1200 ,670);

glVertex2f(-xHealthBarStart, 670);

glEnd();

char temp[40];

glColor3f(0 ,0 ,1);

sprintf(temp,"SCORE = %d",Score);

displayRasterText(-1100 ,600 ,0.4 ,temp);//<---display variable score ?

sprintf(temp," LIFE = %d",alienLife);

displayRasterText(800 ,600 ,0.4 ,temp);

sprintf(temp," LEVEL : %d",GameLvl);

displayRasterText(-100 ,600 ,0.4 ,temp);

glColor3f(1 ,0 ,0);

}

void startScreenDisplay()

{

glLineWidth(50);

SetDisplayMode(MENU\_SCREEN);

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP); //Border

glVertex3f(-750 ,-500 ,0.5);

glVertex3f(-750 ,550 ,0.5);

glVertex3f(750 ,550 ,0.5);

glVertex3f(750 ,-500, 0.5);

glEnd();

glLineWidth(1);

glColor3f(1, 1, 0);

glBegin(GL\_POLYGON); //START GAME PLOYGON

glVertex3f(-200 ,300 ,0.5);

glVertex3f(-200 ,400 ,0.5);

glVertex3f(200 ,400 ,0.5);

glVertex3f(200 ,300, 0.5);

glEnd();

glBegin(GL\_POLYGON); //INSTRUCTIONS POLYGON

glVertex3f(-200, 50 ,0.5);

glVertex3f(-200 ,150 ,0.5);

glVertex3f(200 ,150 ,0.5);

glVertex3f(200 ,50, 0.5);

glEnd();

glBegin(GL\_POLYGON); //QUIT POLYGON

glVertex3f(-200 ,-200 ,0.5);

glVertex3f(-200 ,-100 ,0.5);

glVertex3f(200, -100 ,0.5);

glVertex3f(200, -200 ,0.5);

glEnd();

if(mouseX>=-100 && mouseX<=100 && mouseY>=150 && mouseY<=200){

glColor3f(0 ,0 ,1) ;

if(mButtonPressed){

startGame = true ;

gameOver = false;

mButtonPressed = false;

}

} else

glColor3f(0 , 0, 0);

displayRasterText(-100 ,340 ,0.4 ,"Start Game");

if(mouseX>=-100 && mouseX<=100 && mouseY>=30 && mouseY<=80) {

glColor3f(0 ,0 ,1);

if(mButtonPressed){

instructionsGame = true ;

mButtonPressed = false;

}

} else

glColor3f(0 , 0, 0);

displayRasterText(-120 ,80 ,0.4 ,"Instructions");

if(mouseX>=-100 && mouseX<=100 && mouseY>=-90 && mouseY<=-40){

glColor3f(0 ,0 ,1);

if(mButtonPressed){

gameQuit = true ;

mButtonPressed = false;

}

}

else

glColor3f(0 , 0, 0);

displayRasterText(-100 ,-170 ,0.4 ," Quit");

}

void GameScreenDisplay()

{

SetDisplayMode(GAME\_SCREEN);

DisplayHealthBar();

glScalef(2, 2 ,0);

if(alienLife){

SpaceshipCreate();

}

else {

gameOver=true;

instructionsGame = false;

startScreen = false;

} //<----------------------gameover screen

StoneGenerate();

}

void readFromFile() {

FILE \*fp = fopen("HighScoreFile.txt" ,"r");

int i=0;

if(fp!= NULL){

while(fread(&ch,sizeof(char),1 ,fp)){

highScore[i++] = ch;

}

highScore[i] = '\0';

}

fclose(fp);

}

void writeIntoFile() { //To write high score on to file

FILE \*fp = fopen("HighScoreFile.txt" ,"w");

int i=0;

char temp[40];

if(fp!= NULL){

int n= Score;

while(n){

ch = (n%10)+ '0';

n/=10;

temp[i++] = ch;

}

temp[i] = '\0';

strrev(temp);

puts(temp);

if(temp[0] == '\0')

temp[i++] = '0' ,temp[i++] = '\0';

fwrite(temp ,sizeof(char)\*i ,i ,fp);

}

fclose(fp);

}

void GameOverScreen()

{

SetDisplayMode(MENU\_SCREEN);

glColor3f(0,0,0);

glLineWidth(50);

glBegin(GL\_LINE\_LOOP); //Border

glVertex3f(-650 ,-500 ,0.5);

glVertex3f(-650 ,520 ,0.5);

glVertex3f(650 ,520 ,0.5);

glVertex3f(650 ,-500, 0.5);

glEnd();

glLineWidth(1);

stoneTranslationSpeed=5;

glColor3f(0, 1, 0);

glBegin(GL\_POLYGON); //GAME OVER

glVertex3f(-550 ,810,0.5);

glVertex3f(-550 ,610 ,0.5);

glVertex3f(550 ,610 ,0.5);

glVertex3f(550 ,810, 0.5);

glEnd();

glColor3f(1, 1, 0);

glBegin(GL\_POLYGON); //RESTART POLYGON

glVertex3f(-200, 50 ,0.5);

glVertex3f(-200 ,150 ,0.5);

glVertex3f(200 ,150 ,0.5);

glVertex3f(200 ,50, 0.5);

glEnd();

glBegin(GL\_POLYGON); //QUIT POLYGON

glVertex3f(-200 ,-200 ,0.5);

glVertex3f(-200 ,-100 ,0.5);

glVertex3f(200, -100 ,0.5);

glVertex3f(200, -200 ,0.5);

glEnd();

displayRasterText(-300 ,640 ,0.4 ,"G A M E O V E R ! ! !");

glColor3f(0 , 0, 0);

char temp[40];

sprintf(temp,"Score : %d",Score);

displayRasterText(-100 ,340 ,0.4 ,temp);

readFromFile();

char temp2[40];

if(atoi(highScore) < Score){

writeIntoFile();

sprintf(temp2 ,"Highest Score :%d" ,Score);

} else

sprintf(temp2 ,"Highest Score :%s" ,highScore);

displayRasterText(-250 ,400 ,0.4 ,temp2);

if(mouseX>=-100 && mouseX<=100 && mouseY>=25 && mouseY<=75){

glColor3f(0 ,0 ,1);

if(mButtonPressed){ //Reset game default values

startGame = true ;

gameOver=false;

mButtonPressed = false;

initializeStoneArray();

alienLife=100;

xHealthBarStart=1200;

Score=0;

GameLvl=1;

GameScreenDisplay();

}

} else

glColor3f(0 , 0, 0);

displayRasterText(-70 ,80 ,0.4 ,"Restart");

if(mouseX>=-100 && mouseX<=100 && mouseY>=-100 && mouseY<=-50){

glColor3f(0 ,0 ,1);

if(mButtonPressed){

exit(0);

mButtonPressed = false;

}

}

else

glColor3f(0 , 0, 0);

displayRasterText(-100 ,-170 ,0.4 ," Quit");

}

void StoneGenerate(){

if(xStone[0]>=1200){ //If the last screen hits the end of screen then go to Nxt lvl

GameLvl++;

stoneTranslationSpeed+=3;

Score+=50;

initializeStoneArray();

GameScreenDisplay();

}

for(int i=0; i<MAX\_STONES ;i++){

index = i;

if(mouseX <= (xStone[i]/2+20) && mouseX >=(xStone[i]/2-20) && mouseY >= (yStone[i]/2-20) && mouseY <= (yStone[i]/2+20) && mButtonPressed){

if(stoneAlive[i]){ // IF ALIVE KILL STONE

stoneAlive[i]=0;

Score++;

if(Score%1==0) {

stoneTranslationSpeed+=1; //<--------------Rate of increase of game speed

}

}

}

xStone[i] += stoneTranslationSpeed;

if(stoneAlive[i] ) //stone alive

DrawStone(randomStoneIndices[i]);

}

stoneAngle+=stoneRotationSpeed;

if(stoneAngle > 360) stoneAngle = 0;

}

void backButton() {

if(mouseX <= -450 && mouseX >= -500 && mouseY >= -275 && mouseY <= -250){

glColor3f(0, 0, 1);

if(mButtonPressed){

mButtonPressed = false;

instructionsGame = false;

startScreenDisplay();

}

}

else glColor3f(0, 0, 0);

displayRasterText(-1000 ,-550 ,0, "Back");

}

void InstructionsScreenDisplay()

{

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

SetDisplayMode(MENU\_SCREEN);

//colorBackground();

glColor3f(0, 0, 0);

displayRasterText(-900 ,400 ,0.4 ,"Key 'w' to move up.");

displayRasterText(-900 ,300 ,0.4 ,"Key 's' to move down.");

displayRasterText(-900 ,200 ,0.4 ,"Key 'd' to move right.");

displayRasterText(-900 ,100 ,0.4 ,"Key 'a' to move left.");

displayRasterText(-900 ,0.0 ,0.4 ,"Left mouse click to shoot laser");

//displayRasterText(-900 ,-100 ,0.4 ,"The packet can be placed only when 's' is pressed before.");

displayRasterText(-900 ,-200 ,0.4 ,"You Get 1 point for shooting each objet and 50 points for completing each lvl ");

displayRasterText(-900, -270,0.4,"The Objective is to score maximum points");

backButton();

if(previousScreen)

nextScreen = false ,previousScreen = false; //as set by backButton()

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0,0,1200,700);

if(startGame && !gameOver)

GameScreenDisplay();

else if(instructionsGame)

InstructionsScreenDisplay();

else if(gameOver)

GameOverScreen();

//Make spaceship bigger

else if(startScreen){

startScreenDisplay();

if(gameQuit || startGame || optionsGame || instructionsGame){

//startScreen = false;

if(startGame){

SetDisplayMode(GAME\_SCREEN);

startScreen = false;

} else if(gameQuit)

exit(0);

} else if(instructionsGame) {

SetDisplayMode(GAME\_SCREEN);

InstructionsScreenDisplay();

}

}

//Reset Scaling values

glScalef(1/2 ,1/2 ,0);

glFlush();

glLoadIdentity();

glutSwapBuffers();

}

void somethingMovedRecalculateLaserAngle() {

float mouseXForTan = (-50 - mouseX) + xOne;

float mouseYForTan = (35 - mouseY) + yOne;

float LaserAngleInRadian = atan(mouseYForTan/mouseXForTan);

LaserAngle = (180/PI) \* LaserAngleInRadian;

}

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

{

//if(key=='w' && key=='d' ){xOne+=0.5;yOne+=0.5;}

if(key == 'd') xOne+=SPACESHIP\_SPEED;

if(key == 'a') xOne-=SPACESHIP\_SPEED;

if(key == 'w') {yOne+=SPACESHIP\_SPEED;}

if(key == 's') {yOne-=SPACESHIP\_SPEED;}

if(key == 'd' || key == 'a' || key == 'w' || key == 's')

somethingMovedRecalculateLaserAngle();

display();

}

void myinit()

{

glClearColor(0.5,0.5,0.5,0);

glColor3f(1.0,0.0,0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1200,1200,-700,700); //<-----CHANGE THIS TO GET EXTRA SPACE

// gluOrtho2D(-200,200,-200,200);

glMatrixMode(GL\_MODELVIEW);

}

void passiveMotionFunc(int x,int y) {

//when mouse not clicked

mouseX = float(x)/(m\_viewport[2]/1200.0)-600.0; //converting screen resolution to ortho 2d spec

mouseY = -(float(y)/(m\_viewport[3]/700.0)-350.0);

//Do calculations to find value of LaserAngle

somethingMovedRecalculateLaserAngle();

display();

}

void mouseClick(int buttonPressed ,int state ,int x, int y) {

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

mButtonPressed = true;

else

mButtonPressed = false;

display();

}

void UpdateColorIndexForSpaceshipLights(int value)

{

CI=(CI+1)%3; //Color Index swapping to have rotation effect

display();

glutTimerFunc(250,UpdateColorIndexForSpaceshipLights,0);

}

void idleCallBack() { //when no mouse or keybord pressed

display();

}

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

FILE \*fp = fopen("HighScoreFile.txt" ,"r") ; //check if HighScoreFile.txt exist if not create

if(fp!=NULL)

fclose(fp);

else

writeIntoFile();

glutInit(&argc, argv);

glutInitWindowSize(1200,700);

glutInitWindowPosition(90 ,0);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB);

glutTimerFunc(50,UpdateColorIndexForSpaceshipLights,0);

glutCreateWindow("THE SPACESHIP SHOOTING GAME");

glutDisplayFunc(display);

glutKeyboardFunc(keys);

glutPassiveMotionFunc(passiveMotionFunc);

glBlendFunc(GL\_SRC\_ALPHA ,GL\_ONE\_MINUS\_SRC\_ALPHA);

glutIdleFunc(idleCallBack);

glutMouseFunc(mouseClick);

glGetIntegerv(GL\_VIEWPORT ,m\_viewport);

myinit();

SetDisplayMode(GAME\_SCREEN);

initializeStoneArray();

glutMainLoop();

}

# Máy hơi nước

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

#include <math.h>

#define TRUE 1

#define FALSE 0

/\* Dimensions of texture image. \*/

#define IMAGE\_WIDTH 64

#define IMAGE\_HEIGHT 64

/\* Step to be taken for each rotation. \*/

#define ANGLE\_STEP 10

/\* Magic numbers for relationship b/w cylinder head and crankshaft. \*/

#define MAGNITUDE 120

#define PHASE 270.112

#define FREQ\_DIV 58

#define ARC\_LENGHT 2.7

#define ARC\_RADIUS 0.15

/\* Rotation angles \*/

GLdouble view\_h = 270, view\_v = 0, head\_angle = 0;

GLint crank\_angle = 0;

/\* Crank rotation step. \*/

GLdouble crank\_step = 5;

/\* Toggles \*/

GLshort shaded = TRUE, anim = FALSE;

GLshort texture = FALSE, transparent = FALSE;

GLshort light1 = TRUE, light2 = FALSE;

/\* Storage for the angle look up table and the texture map \*/

GLdouble head\_look\_up\_table[361];

GLubyte image[IMAGE\_WIDTH][IMAGE\_HEIGHT][3];

/\* Indentifiers for each Display list \*/

GLint list\_piston\_shaded = 1;

GLint list\_piston\_texture = 2;

GLint list\_flywheel\_shaded = 4;

GLint list\_flywheel\_texture = 8;

/\* Variable used in the creaton of glu objects \*/

GLUquadricObj \*obj;

/\* Draws a box by scaling a glut cube of size 1. Also checks the shaded

toggle to see which rendering style to use. NB Texture doesn't work

correctly due to the cube being scaled. \*/

void

myBox(GLdouble x, GLdouble y, GLdouble z)

{

glPushMatrix();

glScalef(x, y, z);

if (shaded)

glutSolidCube(1);

else

glutWireCube(1);

glPopMatrix();

}

/\* Draws a cylinder using glu function, drawing flat disc's at each end,

to give the appearence of it being solid. \*/

void

myCylinder(GLUquadricObj \* object, GLdouble outerRadius,

GLdouble innerRadius, GLdouble lenght)

{

glPushMatrix();

gluCylinder(object, outerRadius, outerRadius, lenght, 20, 1);

glPushMatrix();

glRotatef(180, 0.0, 1.0, 0.0);

gluDisk(object, innerRadius, outerRadius, 20, 1);

glPopMatrix();

glTranslatef(0.0, 0.0, lenght);

gluDisk(object, innerRadius, outerRadius, 20, 1);

glPopMatrix();

}

/\* Draws a piston. \*/

void

draw\_piston(void)

{

glPushMatrix();

glColor4f(0.9, 0.6, 0.9, 1.5);

glPushMatrix();

glRotatef(90, 0.0, 1.0, 0.0);

glTranslatef(0.0, 0.0, -0.07);

myCylinder(obj, 0.125, 0.06, 0.12);

glPopMatrix();

glRotatef(-90, 1.0, 0.0, 0.0);

glTranslatef(0.0, 0.0, 0.05);

myCylinder(obj, 0.06, 0.0, 0.6);

glTranslatef(0.0, 0.0, 0.6);

myCylinder(obj, 0.2, 0.0, 0.5);

glPopMatrix();

}

/\* Draws the engine pole and the pivot pole for the cylinder head. \*/

void

draw\_engine\_pole(void)

{

glPushMatrix();

glColor4f(0.9, 0.9, 0.9, 1.0);

myBox(0.5, 3.0, 0.5);

glColor3f(1.5, 2.1, 3.5);

glRotatef(90, 0.0, 1.0, 0.0);

glTranslatef(0.0, 0.9, -0.4);

myCylinder(obj, 0.1, 0.0, 2);

glPopMatrix();

}

/\* Draws the cylinder head at the appropreate angle, doing the necesary

translations for the rotation. \*/

void

draw\_cylinder\_head(void)

{

glPushMatrix();

glColor4f(3.5, 1.0, 0.5, 0.1);

glRotatef(90, 1.0, 0.0, 0.0);

glTranslatef(0, 0.0, 0.4);

glRotatef(head\_angle, 1, 0, 0);

glTranslatef(0, 0.0, -0.4);

myCylinder(obj, 0.23, 0.21, 1.6);

glRotatef(180, 1.0, 0.0, 0.0);

gluDisk(obj, 0, 0.23, 20, 1);

glPopMatrix();

}

/\* Draws the flywheel. \*/

void

draw\_flywheel(void)

{

glPushMatrix();

glColor4f(0.5, 0.5, 1.0, 1.0);

glRotatef(90, 0.0, 1.0, 0.0);

myCylinder(obj, 0.625, 0.08, 0.5);

glPopMatrix();

}

/\* Draws the crank bell, and the pivot pin for the piston. Also calls the

appropreate display list of a piston doing the nesacary rotations before

hand. \*/

void

draw\_crankbell(void)

{

glPushMatrix();

glColor4f(1.0, 2.5, 3.5, 1.0);

glRotatef(90, 0.0, 1.0, 0.0);

myCylinder(obj, 0.3, 0.08, 0.12);

glColor4f(0.5, 0.1, 0.5, 1.0);

glTranslatef(0.0, 0.2, 0.0);

myCylinder(obj, 0.06, 0.0, 0.34);

glTranslatef(0.0, 0.0, 0.22);

glRotatef(90, 0.0, 1.0, 0.0);

glRotatef(crank\_angle - head\_angle, 1.0, 0.0, 0.0);

if (shaded) {

if (texture)

glCallList(list\_piston\_texture);

else

glCallList(list\_piston\_shaded);

} else

draw\_piston();

glPopMatrix();

}

/\* Draws the complete crank. Piston also gets drawn through the crank bell

function. \*/

void

draw\_crank(void)

{

glPushMatrix();

glRotatef(crank\_angle, 1.0, 0.0, 0.0);

glPushMatrix();

glRotatef(90, 0.0, 1.0, 0.0);

glTranslatef(0.0, 0.0, -1.0);

myCylinder(obj, 0.08, 0.0, 1.4);

glPopMatrix();

glPushMatrix();

glTranslatef(0.28, 0.0, 0.0);

draw\_crankbell();

glPopMatrix();

glPushMatrix();

glTranslatef(-0.77, 0.0, 0.0);

if (shaded) {

if (texture)

glCallList(list\_flywheel\_texture);

else

glCallList(list\_flywheel\_shaded);

} else

draw\_flywheel();

glPopMatrix();

glPopMatrix();

}

/\* Main display routine. Clears the drawing buffer and if transparency is

set, displays the model twice, 1st time accepting those fragments with

a ALPHA value of 1 only, then with DEPTH\_BUFFER writing disabled for

those with other values. \*/

void

display(void)

{

int pass;

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

glPushMatrix();

if (transparent) {

glEnable(GL\_ALPHA\_TEST);

pass = 2;

} else {

glDisable(GL\_ALPHA\_TEST);

pass = 0;

}

/\* Rotate the whole model \*/

glRotatef(view\_h, 0, 1, 0);

glRotatef(view\_v, 1, 0, 0);

do {

if (pass == 2) {

glAlphaFunc(GL\_EQUAL, 1);

glDepthMask(GL\_TRUE);

pass--;

} else if (pass != 0) {

glAlphaFunc(GL\_NOTEQUAL, 1);

glDepthMask(GL\_FALSE);

pass--;

}

draw\_engine\_pole();

glPushMatrix();

glTranslatef(0.5, 1.4, 0.0);

draw\_cylinder\_head();

glPopMatrix();

glPushMatrix();

glTranslatef(0.0, -0.8, 0.0);

draw\_crank();

glPopMatrix();

} while (pass > 0);

glDepthMask(GL\_TRUE);

glutSwapBuffers();

glPopMatrix();

}

/\* Called when the window is idle. When called increments the crank angle

by ANGLE\_STEP, updates the head angle and notifies the system that

the screen needs to be updated. \*/

void

animation(void)

{

if ((crank\_angle += crank\_step) >= 360)

crank\_angle = 0;

head\_angle = head\_look\_up\_table[crank\_angle];

glutPostRedisplay();

}

/\* Called when a key is pressed. Checks if it reconises the key and if so

acts on it, updateing the screen. \*/

/\* ARGSUSED1 \*/

void

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

{

switch (key) {

case 's':

if (shaded == FALSE) {

shaded = TRUE;

glShadeModel(GL\_SMOOTH);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

gluQuadricNormals(obj, GLU\_SMOOTH);

gluQuadricDrawStyle(obj, GLU\_FILL);

} else {

shaded = FALSE;

glShadeModel(GL\_FLAT);

glDisable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

glDisable(GL\_COLOR\_MATERIAL);

gluQuadricNormals(obj, GLU\_NONE);

gluQuadricDrawStyle(obj, GLU\_LINE);

gluQuadricTexture(obj, GL\_FALSE);

}

if (texture && !shaded);

else

break;

case 't':

if (texture == FALSE) {

texture = TRUE;

glEnable(GL\_TEXTURE\_2D);

gluQuadricTexture(obj, GL\_TRUE);

} else {

texture = FALSE;

glDisable(GL\_TEXTURE\_2D);

gluQuadricTexture(obj, GL\_FALSE);

}

break;

case 'o':

if (transparent == FALSE) {

transparent = TRUE;

} else {

transparent = FALSE;

}

break;

case 'a':

if ((crank\_angle += crank\_step) >= 360)

crank\_angle = 0;

head\_angle = head\_look\_up\_table[crank\_angle];

break;

case 'z':

if ((crank\_angle -= crank\_step) <= 0)

crank\_angle = 360;

head\_angle = head\_look\_up\_table[crank\_angle];

break;

case '0':

if (light1) {

glDisable(GL\_LIGHT0);

light1 = FALSE;

} else {

glEnable(GL\_LIGHT0);

light1 = TRUE;

}

break;

case '1':

if (light2) {

glDisable(GL\_LIGHT1);

light2 = FALSE;

} else {

glEnable(GL\_LIGHT1);

light2 = TRUE;

}

break;

case '4':

if ((view\_h -= ANGLE\_STEP) <= 0)

view\_h = 360;

break;

case '8':

if ((view\_v += ANGLE\_STEP) >= 360)

view\_v = 0;

break;

case '2':

if ((view\_v -= ANGLE\_STEP) <= 0)

view\_v = 360;

break;

case ' ':

if (anim) {

glutIdleFunc(0);

anim = FALSE;

} else {

glutIdleFunc(animation);

anim = TRUE;

}

break;

case '+':

if ((++crank\_step) > 45)

crank\_step = 45;

break;

case '-':

if ((--crank\_step) <= 0)

crank\_step = 0;

break;

default:

return;

}

glutPostRedisplay();

}

/\* ARGSUSED1 \*/

void

special(int key, int x, int y)

{

switch (key) {

case GLUT\_KEY\_LEFT:

if ((view\_h -= ANGLE\_STEP) <= 0)

view\_h = 360;

break;

case GLUT\_KEY\_RIGHT:

if ((view\_h += ANGLE\_STEP) >= 360)

view\_h = 0;

break;

case GLUT\_KEY\_UP:

if ((view\_v += ANGLE\_STEP) >= 360)

view\_v = 0;

break;

case GLUT\_KEY\_DOWN:

if ((view\_v -= ANGLE\_STEP) <= 0)

view\_v = 360;

break;

default:

return;

}

glutPostRedisplay();

}

/\* Called when a menu option has been selected. Translates the menu item

identifier into a keystroke, then call's the keyboard function. \*/

void

menu(int val)

{

unsigned char key;

switch (val) {

case 1:

key = 's';

break;

case 2:

key = ' ';

break;

case 3:

key = 't';

break;

case 4:

key = 'o';

break;

case 5:

key = '0';

break;

case 7:

key = '+';

break;

case 8:

key = '-';

break;

default:

return;

}

keyboard(key, 0, 0);

}

/\* Initialises the menu of toggles. \*/

void

create\_menu(void)

{

glutCreateMenu(menu);

glutAttachMenu(GLUT\_LEFT\_BUTTON);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutAddMenuEntry("Shaded", 1);

glutAddMenuEntry("Animation", 2);

glutAddMenuEntry("Texture", 3);

glutAddMenuEntry("Transparency", 4);

glutAddMenuEntry("Light On/Off", 5);

glutAddMenuEntry("Speed UP", 7);

glutAddMenuEntry("Slow Down", 8);

}

/\* Makes a simple check pattern image. (Copied from the redbook example

"checker.c".) \*/

void

make\_image(void)

{

int i, j, c;

for (i = 0; i < IMAGE\_WIDTH; i++) {

for (j = 0; j < IMAGE\_HEIGHT; j++) {

c = ((((i & 0x8) == 0) ^ ((j & 0x8)) == 0)) \* 255;

image[i][j][0] = (GLubyte) c;

image[i][j][1] = (GLubyte) c;

image[i][j][2] = (GLubyte) c;

}

}

}

/\* Makes the head look up table for all possible crank angles. \*/

void

make\_table(void)

{

GLint i;

GLdouble k;

for (i = 0, k = 0.0; i < 360; i++, k++) {

head\_look\_up\_table[i] =

MAGNITUDE \* atan(

(ARC\_RADIUS \* sin(PHASE - k / FREQ\_DIV)) /

((ARC\_LENGHT - ARC\_RADIUS \* cos(PHASE - k / FREQ\_DIV))));

}

}

/\* Initialises texturing, lighting, display lists, and everything else

associated with the model. \*/

void

myinit(void)

{

GLfloat mat\_specular[] = {1.0, 1.0, 1.0, 1.0};

GLfloat mat\_shininess[] = {50.0};

GLfloat light\_position1[] = {1.0, 1.0, 1.0, 0.0};

GLfloat light\_position2[] = {-1.0, 1.0, 1.0, 0.0};

glClearColor(0.0, 0.0, 0.0, 0.0);

obj = gluNewQuadric();

make\_table();

make\_image();

/\* Set up Texturing \*/

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, IMAGE\_WIDTH,

IMAGE\_HEIGHT, 0, GL\_RGB, GL\_UNSIGNED\_BYTE,

image);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

/\* Set up Lighting \*/

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position1);

glLightfv(GL\_LIGHT1, GL\_POSITION, light\_position2);

/\* Initial render mode is with full shading and LIGHT 0

enabled. \*/

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glDepthFunc(GL\_LEQUAL);

glEnable(GL\_DEPTH\_TEST);

glDisable(GL\_ALPHA\_TEST);

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE);

glEnable(GL\_COLOR\_MATERIAL);

glShadeModel(GL\_SMOOTH);

/\* Initialise display lists \*/

glNewList(list\_piston\_shaded, GL\_COMPILE);

draw\_piston();

glEndList();

glNewList(list\_flywheel\_shaded, GL\_COMPILE);

draw\_flywheel();

glEndList();

gluQuadricTexture(obj, GL\_TRUE);

glNewList(list\_piston\_texture, GL\_COMPILE);

draw\_piston();

glEndList();

glNewList(list\_flywheel\_texture, GL\_COMPILE);

draw\_flywheel();

glEndList();

gluQuadricTexture(obj, GL\_FALSE);

}

/\* Called when the model's window has been reshaped. \*/

void

myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(65.0, (GLfloat) w / (GLfloat) h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -5.0); /\* viewing transform \*/

glScalef(1.5, 1.5, 1.5);

}

void DrawText(const char \*text, int length, int x, int y)

{

glMatrixMode(GL\_PROJECTION);

double \*matrix = new double[16];

glGetDoublev(GL\_PROJECTION\_MATRIX,matrix);

glLoadIdentity();

glOrtho(0,800,0,600,-5,5);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glPushMatrix();

glLoadIdentity();

glRasterPos2i(x,y);

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

{

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

}

glPopMatrix();

glMatrixMode(GL\_PROJECTION);

glLoadMatrixd(matrix);

glMatrixMode(GL\_MODELVIEW);

}

void display\_other()

{

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

gluLookAt(0,0,-10,0,0,3,0,1,0);

std::string text1;

std::string text2;

std::string text3;

std::string text4;

std::string text5;

std::string text6;

std::string text7;

std::string text8;

std::string text9;

text1 = "Miniature Reciprocating Steam Engine by Sneha Sarkar 1PE16CS156";

text2 = "Keypad Arrow keys rotates object.";

text3 ="Rotate crank: 'a' = anti-clock wise 'z' = clock wise";

text4 ="Crank Speed : '+' = Speed up by 1 '-' = Slow Down by 1";

text5 = "Toggle : 's' = Shading 't' = Texture";

text6 = " : ' ' = Animation 'o' = Transparency";

text7 = " : '0' = Light On/Off";

text8 = " Alternatively a pop up menu with all toggles is attached";

text9 = " to the left mouse button.";

DrawText(text1.data(),text1.size(),200,500);

DrawText(text2.data(),text2.size(),200,460);

DrawText(text3.data(),text3.size(),200,420);

DrawText(text4.data(),text4.size(),200,380);

DrawText(text5.data(),text5.size(),200,340);

DrawText(text6.data(),text6.size(),200,310);

DrawText(text7.data(),text7.size(),200,270);

DrawText(text8.data(),text8.size(),200,230);

DrawText(text9.data(),text9.size(),200,190);

}

/\* Main program. An interactive model of a miniture steam engine.

Sets system in Double Buffered mode and initialises all the call-back

functions. \*/

int

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

{

/\*

puts("Miniature Reciprocating Steam Engine\n");

puts("Keypad Arrow keys (with NUM\_LOCK on) rotates object.");

puts("Rotate crank: 'a' = anti-clock wise 'z' = clock wise");

puts("Crank Speed : '+' = Speed up by 1 '-' = Slow Down by 1");

puts("Toggle : 's' = Shading 't' = Texture");

puts(" : ' ' = Animation 'o' = Transparency");

puts(" : '0' = Right Light '1' = Left Light");

puts(" Alternatively a pop up menu with all toggles is attached");

puts(" to the left mouse button.\n");

\*/

glutInit(&argc, argv);

glutInitWindowSize(800, 800);

glutCreateWindow("Instructions");

glutDisplayFunc(display\_other);

glutInitWindowSize(800, 800);

/\* Transperancy won't work properly without GLUT\_ALPHA \*/

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH | GLUT\_MULTISAMPLE);

glutCreateWindow("Miniature Steam Engine");

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutSpecialFunc(special);

create\_menu();

myinit();

glutReshapeFunc(myReshape);

glutMainLoop();

return 0; /\* ANSI C requires main to return int. \*/

}

# Bóng phản xạ

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

#include <math.h>

#define TRUE 1

#define FALSE 0

/\* Dimensions of texture image. \*/

#define IMAGE\_WIDTH 64

#define IMAGE\_HEIGHT 64

/\* Step to be taken for each rotation. \*/

#define ANGLE\_STEP 10

/\* Magic numbers for relationship b/w cylinder head and crankshaft. \*/

#define MAGNITUDE 120

#define PHASE 270.112

#define FREQ\_DIV 58

#define ARC\_LENGHT 2.7

#define ARC\_RADIUS 0.15

/\* Rotation angles \*/

GLdouble view\_h = 270, view\_v = 0, head\_angle = 0;

GLint crank\_angle = 0;

/\* Crank rotation step. \*/

GLdouble crank\_step = 5;

/\* Toggles \*/

GLshort shaded = TRUE, anim = FALSE;

GLshort texture = FALSE, transparent = FALSE;

GLshort light1 = TRUE, light2 = FALSE;

/\* Storage for the angle look up table and the texture map \*/

GLdouble head\_look\_up\_table[361];

GLubyte image[IMAGE\_WIDTH][IMAGE\_HEIGHT][3];

/\* Indentifiers for each Display list \*/

GLint list\_piston\_shaded = 1;

GLint list\_piston\_texture = 2;

GLint list\_flywheel\_shaded = 4;

GLint list\_flywheel\_texture = 8;

/\* Variable used in the creaton of glu objects \*/

GLUquadricObj \*obj;

/\* Draws a box by scaling a glut cube of size 1. Also checks the shaded

toggle to see which rendering style to use. NB Texture doesn't work

correctly due to the cube being scaled. \*/

void

myBox(GLdouble x, GLdouble y, GLdouble z)

{

glPushMatrix();

glScalef(x, y, z);

if (shaded)

glutSolidCube(1);

else

glutWireCube(1);

glPopMatrix();

}

/\* Draws a cylinder using glu function, drawing flat disc's at each end,

to give the appearence of it being solid. \*/

void

myCylinder(GLUquadricObj \* object, GLdouble outerRadius,

GLdouble innerRadius, GLdouble lenght)

{

glPushMatrix();

gluCylinder(object, outerRadius, outerRadius, lenght, 20, 1);

glPushMatrix();

glRotatef(180, 0.0, 1.0, 0.0);

gluDisk(object, innerRadius, outerRadius, 20, 1);

glPopMatrix();

glTranslatef(0.0, 0.0, lenght);

gluDisk(object, innerRadius, outerRadius, 20, 1);

glPopMatrix();

}

/\* Draws a piston. \*/

void

draw\_piston(void)

{

glPushMatrix();

glColor4f(0.9, 0.6, 0.9, 1.5);

glPushMatrix();

glRotatef(90, 0.0, 1.0, 0.0);

glTranslatef(0.0, 0.0, -0.07);

myCylinder(obj, 0.125, 0.06, 0.12);

glPopMatrix();

glRotatef(-90, 1.0, 0.0, 0.0);

glTranslatef(0.0, 0.0, 0.05);

myCylinder(obj, 0.06, 0.0, 0.6);

glTranslatef(0.0, 0.0, 0.6);

myCylinder(obj, 0.2, 0.0, 0.5);

glPopMatrix();

}

/\* Draws the engine pole and the pivot pole for the cylinder head. \*/

void

draw\_engine\_pole(void)

{

glPushMatrix();

glColor4f(0.9, 0.9, 0.9, 1.0);

myBox(0.5, 3.0, 0.5);

glColor3f(1.5, 2.1, 3.5);

glRotatef(90, 0.0, 1.0, 0.0);

glTranslatef(0.0, 0.9, -0.4);

myCylinder(obj, 0.1, 0.0, 2);

glPopMatrix();

}

/\* Draws the cylinder head at the appropreate angle, doing the necesary

translations for the rotation. \*/

void

draw\_cylinder\_head(void)

{

glPushMatrix();

glColor4f(3.5, 1.0, 0.5, 0.1);

glRotatef(90, 1.0, 0.0, 0.0);

glTranslatef(0, 0.0, 0.4);

glRotatef(head\_angle, 1, 0, 0);

glTranslatef(0, 0.0, -0.4);

myCylinder(obj, 0.23, 0.21, 1.6);

glRotatef(180, 1.0, 0.0, 0.0);

gluDisk(obj, 0, 0.23, 20, 1);

glPopMatrix();

}

/\* Draws the flywheel. \*/

void

draw\_flywheel(void)

{

glPushMatrix();

glColor4f(0.5, 0.5, 1.0, 1.0);

glRotatef(90, 0.0, 1.0, 0.0);

myCylinder(obj, 0.625, 0.08, 0.5);

glPopMatrix();

}

/\* Draws the crank bell, and the pivot pin for the piston. Also calls the

appropreate display list of a piston doing the nesacary rotations before

hand. \*/

void

draw\_crankbell(void)

{

glPushMatrix();

glColor4f(1.0, 2.5, 3.5, 1.0);

glRotatef(90, 0.0, 1.0, 0.0);

myCylinder(obj, 0.3, 0.08, 0.12);

glColor4f(0.5, 0.1, 0.5, 1.0);

glTranslatef(0.0, 0.2, 0.0);

myCylinder(obj, 0.06, 0.0, 0.34);

glTranslatef(0.0, 0.0, 0.22);

glRotatef(90, 0.0, 1.0, 0.0);

glRotatef(crank\_angle - head\_angle, 1.0, 0.0, 0.0);

if (shaded) {

if (texture)

glCallList(list\_piston\_texture);

else

glCallList(list\_piston\_shaded);

} else

draw\_piston();

glPopMatrix();

}

/\* Draws the complete crank. Piston also gets drawn through the crank bell

function. \*/

void

draw\_crank(void)

{

glPushMatrix();

glRotatef(crank\_angle, 1.0, 0.0, 0.0);

glPushMatrix();

glRotatef(90, 0.0, 1.0, 0.0);

glTranslatef(0.0, 0.0, -1.0);

myCylinder(obj, 0.08, 0.0, 1.4);

glPopMatrix();

glPushMatrix();

glTranslatef(0.28, 0.0, 0.0);

draw\_crankbell();

glPopMatrix();

glPushMatrix();

glTranslatef(-0.77, 0.0, 0.0);

if (shaded) {

if (texture)

glCallList(list\_flywheel\_texture);

else

glCallList(list\_flywheel\_shaded);

} else

draw\_flywheel();

glPopMatrix();

glPopMatrix();

}

/\* Main display routine. Clears the drawing buffer and if transparency is

set, displays the model twice, 1st time accepting those fragments with

a ALPHA value of 1 only, then with DEPTH\_BUFFER writing disabled for

those with other values. \*/

void

display(void)

{

int pass;

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

glPushMatrix();

if (transparent) {

glEnable(GL\_ALPHA\_TEST);

pass = 2;

} else {

glDisable(GL\_ALPHA\_TEST);

pass = 0;

}

/\* Rotate the whole model \*/

glRotatef(view\_h, 0, 1, 0);

glRotatef(view\_v, 1, 0, 0);

do {

if (pass == 2) {

glAlphaFunc(GL\_EQUAL, 1);

glDepthMask(GL\_TRUE);

pass--;

} else if (pass != 0) {

glAlphaFunc(GL\_NOTEQUAL, 1);

glDepthMask(GL\_FALSE);

pass--;

}

draw\_engine\_pole();

glPushMatrix();

glTranslatef(0.5, 1.4, 0.0);

draw\_cylinder\_head();

glPopMatrix();

glPushMatrix();

glTranslatef(0.0, -0.8, 0.0);

draw\_crank();

glPopMatrix();

} while (pass > 0);

glDepthMask(GL\_TRUE);

glutSwapBuffers();

glPopMatrix();

}

/\* Called when the window is idle. When called increments the crank angle

by ANGLE\_STEP, updates the head angle and notifies the system that

the screen needs to be updated. \*/

void

animation(void)

{

if ((crank\_angle += crank\_step) >= 360)

crank\_angle = 0;

head\_angle = head\_look\_up\_table[crank\_angle];

glutPostRedisplay();

}

/\* Called when a key is pressed. Checks if it reconises the key and if so

acts on it, updateing the screen. \*/

/\* ARGSUSED1 \*/

void

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

{

switch (key) {

case 's':

if (shaded == FALSE) {

shaded = TRUE;

glShadeModel(GL\_SMOOTH);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

gluQuadricNormals(obj, GLU\_SMOOTH);

gluQuadricDrawStyle(obj, GLU\_FILL);

} else {

shaded = FALSE;

glShadeModel(GL\_FLAT);

glDisable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

glDisable(GL\_COLOR\_MATERIAL);

gluQuadricNormals(obj, GLU\_NONE);

gluQuadricDrawStyle(obj, GLU\_LINE);

gluQuadricTexture(obj, GL\_FALSE);

}

if (texture && !shaded);

else

break;

case 't':

if (texture == FALSE) {

texture = TRUE;

glEnable(GL\_TEXTURE\_2D);

gluQuadricTexture(obj, GL\_TRUE);

} else {

texture = FALSE;

glDisable(GL\_TEXTURE\_2D);

gluQuadricTexture(obj, GL\_FALSE);

}

break;

case 'o':

if (transparent == FALSE) {

transparent = TRUE;

} else {

transparent = FALSE;

}

break;

case 'a':

if ((crank\_angle += crank\_step) >= 360)

crank\_angle = 0;

head\_angle = head\_look\_up\_table[crank\_angle];

break;

case 'z':

if ((crank\_angle -= crank\_step) <= 0)

crank\_angle = 360;

head\_angle = head\_look\_up\_table[crank\_angle];

break;

case '0':

if (light1) {

glDisable(GL\_LIGHT0);

light1 = FALSE;

} else {

glEnable(GL\_LIGHT0);

light1 = TRUE;

}

break;

case '1':

if (light2) {

glDisable(GL\_LIGHT1);

light2 = FALSE;

} else {

glEnable(GL\_LIGHT1);

light2 = TRUE;

}

break;

case '4':

if ((view\_h -= ANGLE\_STEP) <= 0)

view\_h = 360;

break;

case '8':

if ((view\_v += ANGLE\_STEP) >= 360)

view\_v = 0;

break;

case '2':

if ((view\_v -= ANGLE\_STEP) <= 0)

view\_v = 360;

break;

case ' ':

if (anim) {

glutIdleFunc(0);

anim = FALSE;

} else {

glutIdleFunc(animation);

anim = TRUE;

}

break;

case '+':

if ((++crank\_step) > 45)

crank\_step = 45;

break;

case '-':

if ((--crank\_step) <= 0)

crank\_step = 0;

break;

default:

return;

}

glutPostRedisplay();

}

/\* ARGSUSED1 \*/

void

special(int key, int x, int y)

{

switch (key) {

case GLUT\_KEY\_LEFT:

if ((view\_h -= ANGLE\_STEP) <= 0)

view\_h = 360;

break;

case GLUT\_KEY\_RIGHT:

if ((view\_h += ANGLE\_STEP) >= 360)

view\_h = 0;

break;

case GLUT\_KEY\_UP:

if ((view\_v += ANGLE\_STEP) >= 360)

view\_v = 0;

break;

case GLUT\_KEY\_DOWN:

if ((view\_v -= ANGLE\_STEP) <= 0)

view\_v = 360;

break;

default:

return;

}

glutPostRedisplay();

}

/\* Called when a menu option has been selected. Translates the menu item

identifier into a keystroke, then call's the keyboard function. \*/

void

menu(int val)

{

unsigned char key;

switch (val) {

case 1:

key = 's';

break;

case 2:

key = ' ';

break;

case 3:

key = 't';

break;

case 4:

key = 'o';

break;

case 5:

key = '0';

break;

case 7:

key = '+';

break;

case 8:

key = '-';

break;

default:

return;

}

keyboard(key, 0, 0);

}

/\* Initialises the menu of toggles. \*/

void

create\_menu(void)

{

glutCreateMenu(menu);

glutAttachMenu(GLUT\_LEFT\_BUTTON);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutAddMenuEntry("Shaded", 1);

glutAddMenuEntry("Animation", 2);

glutAddMenuEntry("Texture", 3);

glutAddMenuEntry("Transparency", 4);

glutAddMenuEntry("Light On/Off", 5);

glutAddMenuEntry("Speed UP", 7);

glutAddMenuEntry("Slow Down", 8);

}

/\* Makes a simple check pattern image. (Copied from the redbook example

"checker.c".) \*/

void

make\_image(void)

{

int i, j, c;

for (i = 0; i < IMAGE\_WIDTH; i++) {

for (j = 0; j < IMAGE\_HEIGHT; j++) {

c = ((((i & 0x8) == 0) ^ ((j & 0x8)) == 0)) \* 255;

image[i][j][0] = (GLubyte) c;

image[i][j][1] = (GLubyte) c;

image[i][j][2] = (GLubyte) c;

}

}

}

/\* Makes the head look up table for all possible crank angles. \*/

void

make\_table(void)

{

GLint i;

GLdouble k;

for (i = 0, k = 0.0; i < 360; i++, k++) {

head\_look\_up\_table[i] =

MAGNITUDE \* atan(

(ARC\_RADIUS \* sin(PHASE - k / FREQ\_DIV)) /

((ARC\_LENGHT - ARC\_RADIUS \* cos(PHASE - k / FREQ\_DIV))));

}

}

/\* Initialises texturing, lighting, display lists, and everything else

associated with the model. \*/

void

myinit(void)

{

GLfloat mat\_specular[] = {1.0, 1.0, 1.0, 1.0};

GLfloat mat\_shininess[] = {50.0};

GLfloat light\_position1[] = {1.0, 1.0, 1.0, 0.0};

GLfloat light\_position2[] = {-1.0, 1.0, 1.0, 0.0};

glClearColor(0.0, 0.0, 0.0, 0.0);

obj = gluNewQuadric();

make\_table();

make\_image();

/\* Set up Texturing \*/

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, IMAGE\_WIDTH,

IMAGE\_HEIGHT, 0, GL\_RGB, GL\_UNSIGNED\_BYTE,

image);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

/\* Set up Lighting \*/

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position1);

glLightfv(GL\_LIGHT1, GL\_POSITION, light\_position2);

/\* Initial render mode is with full shading and LIGHT 0

enabled. \*/

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glDepthFunc(GL\_LEQUAL);

glEnable(GL\_DEPTH\_TEST);

glDisable(GL\_ALPHA\_TEST);

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE);

glEnable(GL\_COLOR\_MATERIAL);

glShadeModel(GL\_SMOOTH);

/\* Initialise display lists \*/

glNewList(list\_piston\_shaded, GL\_COMPILE);

draw\_piston();

glEndList();

glNewList(list\_flywheel\_shaded, GL\_COMPILE);

draw\_flywheel();

glEndList();

gluQuadricTexture(obj, GL\_TRUE);

glNewList(list\_piston\_texture, GL\_COMPILE);

draw\_piston();

glEndList();

glNewList(list\_flywheel\_texture, GL\_COMPILE);

draw\_flywheel();

glEndList();

gluQuadricTexture(obj, GL\_FALSE);

}

/\* Called when the model's window has been reshaped. \*/

void

myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(65.0, (GLfloat) w / (GLfloat) h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -5.0); /\* viewing transform \*/

glScalef(1.5, 1.5, 1.5);

}

void DrawText(const char \*text, int length, int x, int y)

{

glMatrixMode(GL\_PROJECTION);

double \*matrix = new double[16];

glGetDoublev(GL\_PROJECTION\_MATRIX,matrix);

glLoadIdentity();

glOrtho(0,800,0,600,-5,5);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glPushMatrix();

glLoadIdentity();

glRasterPos2i(x,y);

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

{

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

}

glPopMatrix();

glMatrixMode(GL\_PROJECTION);

glLoadMatrixd(matrix);

glMatrixMode(GL\_MODELVIEW);

}

void display\_other()

{

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

gluLookAt(0,0,-10,0,0,3,0,1,0);

std::string text1;

std::string text2;

std::string text3;

std::string text4;

std::string text5;

std::string text6;

std::string text7;

std::string text8;

std::string text9;

text1 = "Miniature Reciprocating Steam Engine by Sneha Sarkar 1PE16CS156";

text2 = "Keypad Arrow keys rotates object.";

text3 ="Rotate crank: 'a' = anti-clock wise 'z' = clock wise";

text4 ="Crank Speed : '+' = Speed up by 1 '-' = Slow Down by 1";

text5 = "Toggle : 's' = Shading 't' = Texture";

text6 = " : ' ' = Animation 'o' = Transparency";

text7 = " : '0' = Light On/Off";

text8 = " Alternatively a pop up menu with all toggles is attached";

text9 = " to the left mouse button.";

DrawText(text1.data(),text1.size(),200,500);

DrawText(text2.data(),text2.size(),200,460);

DrawText(text3.data(),text3.size(),200,420);

DrawText(text4.data(),text4.size(),200,380);

DrawText(text5.data(),text5.size(),200,340);

DrawText(text6.data(),text6.size(),200,310);

DrawText(text7.data(),text7.size(),200,270);

DrawText(text8.data(),text8.size(),200,230);

DrawText(text9.data(),text9.size(),200,190);

}

/\* Main program. An interactive model of a miniture steam engine.

Sets system in Double Buffered mode and initialises all the call-back

functions. \*/

int

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

{

/\*

puts("Miniature Reciprocating Steam Engine\n");

puts("Keypad Arrow keys (with NUM\_LOCK on) rotates object.");

puts("Rotate crank: 'a' = anti-clock wise 'z' = clock wise");

puts("Crank Speed : '+' = Speed up by 1 '-' = Slow Down by 1");

puts("Toggle : 's' = Shading 't' = Texture");

puts(" : ' ' = Animation 'o' = Transparency");

puts(" : '0' = Right Light '1' = Left Light");

puts(" Alternatively a pop up menu with all toggles is attached");

puts(" to the left mouse button.\n");

\*/

glutInit(&argc, argv);

glutInitWindowSize(800, 800);

glutCreateWindow("Instructions");

glutDisplayFunc(display\_other);

glutInitWindowSize(800, 800);

/\* Transperancy won't work properly without GLUT\_ALPHA \*/

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH | GLUT\_MULTISAMPLE);

glutCreateWindow("Miniature Steam Engine");

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutSpecialFunc(special);

create\_menu();

myinit();

glutReshapeFunc(myReshape);

glutMainLoop();

return 0; /\* ANSI C requires main to return int. \*/

}

# Phóng cảnh nhà ngày đêm

////////////////////////////////////////////////////////

// Abdelhak Mehadjbia

// opengl house with c language 2d hadler

////////////////////////////////////////////////////////

// library opengl

#include <GL/glut.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define M\_PI 3.14159265358979323846

float ballX = -0.8f;

float ballY = -0.0f;

float ballZ = -1.2f;

int night=0;

static int flag=1;

// function for initialise the world

double size=150;

void initDay (void)

{

glClearColor(0.60,0.40,0.12, 1.0);

glViewport(0,0,500,500);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, size, 0.0, size);

}

// function for drawing a square

void draw\_square() {

glBegin(GL\_QUADS);

glVertex2f(-0.5f, -0.5f);

glVertex2f( 0.5f, -0.5f);

glVertex2f( 0.5f, 0.5f);

glVertex2f(-0.5f, 0.5f);

glEnd();

}

// drawing the ball

void drawBall(void) {

glColor3ub(255,255,0); //set ball colour

glTranslatef(ballX,ballY,ballZ); //moving it toward the screen a bit on creation

glutSolidSphere (0.25, 30, 30); //create ball.

}

// drawing triangle

void draw\_triangle() {

glBegin(GL\_TRIANGLES);

glVertex2f(-0.5f, -0.5f);

glVertex2f(+0.5f, -0.5f);

glVertex2f( 0.0f, 0.5f);

glEnd();

}

// the main function of our design

void display(void)

{

// if we press n in keyboard will display this

if (night==1)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0,0.0,1.0,0.0);

glLineWidth(1);

glPushMatrix();

glutWireCube (1.0);

// the stars

// star 1

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(100.5, 142.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(100.5, 142.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(100.5, 142.5, 1.0);

glRotatef(-290, 0.0, 0.0, 1.0);

glScalef(14, 6, 0.0);

draw\_triangle();

glPopMatrix();

// star 2

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(60.5, 142.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(60.5, 142.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(60.5, 142.5, 1.0);

glRotatef(-290, 0.0, 0.0, 1.0);

glScalef(14, 6, 0.0);

draw\_triangle();

glPopMatrix();

// star 3

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(50.5, 127.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(50.5, 127.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(50.5, 127.5, 1.0);

glRotatef(-290, 0.0, 0.0, 1.0);

glScalef(14, 6, 0.0);

draw\_triangle();

glPopMatrix();

// star 4

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(130.5, 142.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(130.5, 142.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(130.5, 142.5, 1.0);

glRotatef(-290, 0.0, 0.0, 1.0);

glScalef(14, 6, 0.0);

draw\_triangle();

glPopMatrix();

// star 5

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(136.5, 120.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(136.5, 120.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(136.5, 120.5, 1.0);

glRotatef(-290, 0.0, 0.0, 1.0);

glScalef(14, 6, 0.0);

draw\_triangle();

glPopMatrix();

// star 6

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(115.5, 115.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(115.5, 115.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(15, 6, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.0f, 0.0f);

glTranslatef(115.5, 115.5, 1.0);

glRotatef(-290, 0.0, 0.0, 1.0);

glScalef(14, 6, 0.0);

draw\_triangle();

glPopMatrix();

// the rest of design

// the home

glPushMatrix();

glColor3f(0.0,1.0,0.0);

glTranslatef(40,35.5,1);

glScalef(40, 63, 1);

draw\_square();

glPopMatrix();

//roof of the house

glPushMatrix();

glColor3f(0.0,0.9,1.0);

glTranslatef(40,82,1);

glScalef(40, 30, 1);

draw\_triangle();

glPopMatrix();

// the house door

glPushMatrix();

glColor3f(1.0,0.0,0.0);

glTranslatef(40,15.5,1);

glScalef(10, 22, 1);

draw\_square();

glPopMatrix();

//

//

// the left house window

glPushMatrix();

glColor3f(1.0,1.0,1.0);

glTranslatef(30,46,1);

glScalef(10, 10, 1);

draw\_square();

glPopMatrix();

//

// left window's bar

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(25, 46);

glVertex2f(35, 46);

glEnd();

glPopMatrix();

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(30, 51);

glVertex2f(30, 41);

glEnd();

glPopMatrix();

//

// right house's window

glPushMatrix();

glColor3f(1.0,1.0,1.0);

glTranslatef(50,46,1);

glScalef(10, 10, 1);

draw\_square();

glPopMatrix();

//

//right window's bar

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(45, 46);

glVertex2f(55, 46);

glEnd();

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(50, 51);

glVertex2f(50, 41);

glEnd();

glPopMatrix();

//

//the tree

glPushMatrix();

glColor3f(0.0, 1.0, 0.0);

glTranslatef(125, 19.0, 1.0);

glScalef(5, 31.3, 0.0);

draw\_square();

glPopMatrix();

// the leaves of the tree

glPushMatrix();

glColor3f(0.0, 1.0, 0.0);

glTranslatef(125.5, 37.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(25, 7, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0, 1.0, 0.0);

glTranslatef(124.1, 37.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(25, 7, 0.0);

draw\_triangle();

glPopMatrix();

//

// the ground

glLineWidth(10);

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(0.0, 4);

glVertex2f(130, 4);

glEnd();

glPopMatrix();

//

glPopMatrix();

glFlush();

}

else {

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.60,0.40,0.12, 1.0);

glLineWidth(1);

glPushMatrix();

glutWireCube (1.0);

//the ball

glPushMatrix();

glTranslatef(30,80,1);

glScalef(40, 63, 1);

drawBall();

glPopMatrix();

// the home

glPushMatrix();

glColor3f(0.0,1.0,0.0);

glTranslatef(40,35.5,1);

glScalef(40, 63, 1);

draw\_square();

glPopMatrix();

//roof of the house

glPushMatrix();

glColor3f(0.0,0.9,1.0);

glTranslatef(40,82,1);

glScalef(40, 30, 1);

draw\_triangle();

glPopMatrix();

// the house door

glPushMatrix();

glColor3f(1.0,0.0,0.0);

glTranslatef(40,15.5,1);

glScalef(10, 22, 1);

draw\_square();

glPopMatrix();

//

//

// the left house window

glPushMatrix();

glColor3f(1.0,1.0,1.0);

glTranslatef(30,46,1);

glScalef(10, 10, 1);

draw\_square();

glPopMatrix();

//

// left window's bar

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(25, 46);

glVertex2f(35, 46);

glEnd();

glPopMatrix();

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(30, 51);

glVertex2f(30, 41);

glEnd();

glPopMatrix();

//

// right house's window

glPushMatrix();

glColor3f(1.0,1.0,1.0);

glTranslatef(50,46,1);

glScalef(10, 10, 1);

draw\_square();

glPopMatrix();

//

//right window's bar

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(45, 46);

glVertex2f(55, 46);

glEnd();

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex2f(50, 51);

glVertex2f(50, 41);

glEnd();

glPopMatrix();

//

//the tree

glPushMatrix();

glColor3f(0.0, 1.0, 0.0);

glTranslatef(125, 19.0, 1.0);

glScalef(5, 31.3, 0.0);

draw\_square();

glPopMatrix();

// the leaves of the tree

glPushMatrix();

glColor3f(0.0, 1.0, 0.0);

glTranslatef(125.5, 37.5, 1.0);

glRotatef(-150, 0.0, 0.0, 1.0);

glScalef(25, 7, 0.0);

draw\_triangle();

glPopMatrix();

glPushMatrix();

glColor3f(0.0, 1.0, 0.0);

glTranslatef(124.1, 37.5, 1.0);

glRotatef(150, 0.0, 0.0, 1.0);

glScalef(25, 7, 0.0);

draw\_triangle();

glPopMatrix();

//

// the ground

glLineWidth(10);

glPushMatrix();

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0,0.0,0.0);

glVertex3f(0.0, 4,-1);

glVertex3f(130, 4,-1);

glEnd();

glPopMatrix();

//

glPopMatrix();

glFlush();

}}

void displayall(){

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective( 90.0,0.8,-1.0,1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(150,150,150,0,0,0,0,1,0);

display();

}

// function to recalculate the scene

static void idle(void)

{

glutPostRedisplay();

}

// reshape function to resize the viewport

void reshape (int w,int h)

{

if(w >=h) glViewport(0,0,(GLsizei)h, (GLsizei)h);

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

}

void keyPress(int key,int x, int y)

{

switch (key)

{

case GLUT\_KEY\_RIGHT:glutFullScreen();

case GLUT\_KEY\_DOWN: size++;

glLoadIdentity(); gluOrtho2D(0.0, size, 0.0, size); break;

case GLUT\_KEY\_UP: size--;

glLoadIdentity(); gluOrtho2D(0.0, size, 0.0, size); break;

default: break;

}

glutPostRedisplay();

}

void keyPress2(unsigned char key,int x, int y)

{

switch (key)

{

case 'n': if (night==0)

{

night=1;

}else night=0;

break;

case 27 : exit(0);

default: break;

}

glutPostRedisplay();

}

//float \_angle = 30.0f;

void update(int value) {

if(ballX>0.9f)

{

ballX = -0.8f;

ballY = -0.0f;

flag=1;

}

if(flag)

{

ballX += 0.008f;

ballY +=0.001f;

if(ballX>0.01)

{

flag=0;

}

}

if (!flag)

{

ballX += 0.006f;

ballY -=0.0009f;

if(ballX<-0.7)

{

flag=1;

}

}

glutPostRedisplay(); //Tell GLUT that the display has changed

//Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(15, update, 0);

}

int

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

{

glutInit(&argc,argv);

int mode = GLUT\_RGB|GLUT\_SINGLE;

glutInitDisplayMode(mode);

glutInitWindowSize(500,500);

glutCreateWindow("Lab 2");

glutReshapeFunc(reshape);

glutDisplayFunc(display);

/\*glutFullScreen();\*/

glutSpecialFunc(keyPress);

glutKeyboardFunc(keyPress2);

initDay();

glutTimerFunc(50, update, 0);

glutMainLoop();

return 0;

}