#include<Windows.h>

#include<GL/glut.h>

#include<stdlib.h>

#include<math.h>

#include<conio.h>

#include<stdio.h>

#include<iostream>

#include<iomanip>

#include<gl/glut.h>

using namespace std;

/\*

This is an example of a 2 dimensional annimation. Two icons, a square and triangle are loaded. Then the program utilizes

the MODELVIEW Matrix to move and rotate the figures before putting them into the graphics pipeline for rendering.

The program also utilizes a small annimation driver found in TimerFunction. This driver changes the global variables

(theta, dx and dy for the square);(thatat, dxt, and dyt for the triangle) before pushing the icons through the MODELVIEW

MATRIX.

//\*\*\*\*\*\*\*\*\*\*\* Global values\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* These values are global because the timing call back functions will only take certain parameters

hence their needs to be global variables to communicate with these functions \*/

float theta = 0.0;//global angular value for rotationn

float scale1 = 1.0;//global scaling value for square

float dx = 7.0, dy = -3.0;//global movement value for dx and dy/

float thetaw = 0.0, zetaw = 0.0, dxw = -7.0, dyw = -3.0;// global values for the triangle

int frame = 1;

int count = 0;

void init(void);//this is a function to initialize the window clear color

void RenderScene(void);//this is a function to draw a square in an opened window

void loadicon(float[][5], float[][5], float[], float[], float[], float[], float[], float[], float[], float[]); //Load the polyman icon

void drawicon(float[][5], float[][5], float[], float[], float[], float[], float[], float[], float[], float[]); //Draw the icon the two first float for the square and the others for the line

void settrans3(void);/\*sets the rotation/translation matrix the MODELVIEW MATRIX for the triangle\*/

void settrans2(void);/\* sets the rotation/translation matrix the MODELVIEW MATRIX for the square\*/

void loadicon2(float[][5], float[][5], float[], float[], float[], float[], float[], float[], float[], float[]); //Load the polyman icon

void drawicon2(float[][5], float[][5], float[], float[], float[], float[], float[], float[], float[], float[]); //Draw the icon the two first float for the square and the others for the line

void SetupRC(void);//sets up the clear color

void TimerFunction(int);//this call back function is call each 30 ms and changes the location,scale and rotation

// of the square.

//Main Program

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

{//set up window title

char header[] = "Wireframe Polyman by Chris Stewart";

/\*glutInit() initializes GLUT. Takes the command line arguments which are used to initialize the native

window system. This function must be called before any other GLUT functions. \*/

glutInit(&argc, argv);

// Set up the display mode with a single buffer and RGB colors

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

//Initialize window size and position

glutInitWindowSize(560, 440);

glutInitWindowPosition(140, 20);

//Initialize background color in window to red

SetupRC();

// Open and Label Window

glutCreateWindow(header);

glutDisplayFunc(RenderScene);

glutTimerFunc(30, TimerFunction, 1);

//Now draw the scene

glutMainLoop();

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RenderScene Function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RenderScene(void)

{

float xdel = 0.25;

float px[7][5], py[7][5],pz[2], plx[4], ply[4], plz[4], pl2x[4], pl2y[4], pl3x[3], pl3y[3], pl4x[4], pl4y[4], eye[2];// These variables hold the pattern for the icon square plus line

float pxw[7][5], pyw[7][5], pzw[2], plxw[4], plyw[4], plyz[4], pl2xw[4], pl2yw[4], pl3xw[3], pl3yw[3], pl4xw[4], pl4yw[4], eyew[2];// These variables hold the pattern for the icon square plus line

float pxt1[3], pyt1[3], pxt2[3], pyt2[3];/\*these variables hold the pattern for the triangle \*/

//clear the window with the current background color

cout << "in RenderScene" << endl;

//set the current drawing color to white

glColor3f(1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

//set the viewport to the window dimensions

glViewport(0, 0, 540, 440);

//Establish the clipping volume in user coordinates

glOrtho(-7.0, 7.0, -7.0, 7.0, 5.0, -5.0);

loadicon(px, py,pz, plx, ply, pl2x, pl2y, pl3x, pl3y,eye);

loadicon2(pxw, pyw, pz, plxw, plyw, pl2xw, pl2yw, pl3xw, pl3yw, eyew);

//glEnable(GL\_DEPTH\_TEST);

/\* draw the i

con untransformed \*/

// Clear the window with the background color

glClear(GL\_COLOR\_BUFFER\_BIT);

//set the current drawing color to white

glColor3f(1.0, 1.0, 1.0);

//Set the MODELVIEW MATRIX for the Square

// settrans2();

/\*now draw the square. Note that in the call to drawicon below, we use the values

returned from loadicon (px,py,plx,ply.pxt1,pyt1,plt2,pyt2). No need to transform

them, the MODELVIEW MATRIX set in settrans2() will transform them \*/

//drawicon(px,py, plx,ply);

// Now Set the MODELVIEW MATRIX for the Triangle

settrans3();

/\*Now Draw the Triangle again note the direct use of pxt1,pyt1,pxt2,pyt2 from loadicon. Again

we depend on the MODELVIEW MATRIX set in settrans3() to transform the triangle pattern \*/

drawicon2(pxw, pyw, pz, plxw, plyw, pl2xw, pl2yw, pl3xw, pl3yw, eyew);

settrans2();

drawicon(px, py, pz, plx, ply, pl2x, pl2y, pl3x, pl3y, eye);

glEnd();

glutSwapBuffers();

return;

};//end of render scene

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Load Icon Function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loadicon(float px[][5], float py[][5], float pz[2], float plx[], float ply[], float pl2x[], float pl2y[], float pl3x[], float pl3y[], float eye[]) //Loads the polyman

{

if (dx == 0) {

px[0][0] = -1.125; py[0][0] = 0.0;

px[0][1] = -0.625; py[0][1] = 0.75;

px[0][2] = 0.625; py[0][2] = 0.75;

px[0][3] = 1.125; py[0][3] = -0;

px[0][4] = -1.125; py[0][4] = 0.0;

px[1][0] = -0.375; py[1][0] = 0.0;

px[1][1] = 1.125; py[1][1] = -0;

px[1][2] = 0.625; py[1][2] = -0.75;

px[1][3] = -0.625; py[1][3] = -0.75;

px[1][4] = -0.375; py[1][4] = 0.0;

}

else {

//Set the coordinates of the body

px[0][0] = -1.125; py[0][0] = 0.0;

px[0][1] = -0.625; py[0][1] = 0.75;

px[0][2] = 0.625; py[0][2] = 0.75;

px[0][3] = 1.125; py[0][3] = -0;

px[0][4] = -1.125; py[0][4] = 0.0;

px[1][0] = -1.125; py[1][0] = 0.0;

px[1][1] = 1.125; py[1][1] = -0;

px[1][2] = 0.625; py[1][2] = -0.75;

px[1][3] = -0.625; py[1][3] = -0.75;

px[1][4] = -1.125; py[1][4] = 0.0;

}

//set the right foot

plx[0] = -0.25; ply[0] = -0.5;

plx[1] = -0.25; ply[1] = -1.0;

plx[2] = -0.50; ply[2] = -1.0;

plx[3] = -0.25; ply[3] = -1.0;

//set the left foot

pl2x[0] = 0.25; pl2y[0] = -0.5;

pl2x[1] = 0.25; pl2y[1] = -1.0;

pl2x[2] = 0.0; pl2y[2] = -1.0;

pl2x[3] = 0.25; pl2y[3] = -1.0;

//set the Closed mouth

pl3x[0] = -0.375; pl3y[0] = -0.0;

pl3x[1] = -0.875; pl3y[1] = -0.5;

pl3x[2] = -0.875; pl3y[2] = -0.5;

//set the eye

eye[0] = -0.5; eye[1] = 0.5;

pz[0] = 0.5;

pz[1] = -0.5;

return;

} //end of loadicon

void loadicon2(float px[][5], float py[][5], float pz[2], float plx[], float ply[], float pl2x[], float pl2y[], float pl3x[], float pl3y[], float eye[]) //Loads the polyman

{

if (dy > -2.0) {

px[0][0] = -1.125; py[0][0] = 0.0;

px[0][1] = -0.625; py[0][1] = 0.75;

px[0][2] = 0.625; py[0][2] = 0.75;

px[0][3] = 1.125; py[0][3] = -0;

px[0][4] = -1.125; py[0][4] = 0.0;

px[1][0] = -0.375; py[1][0] = 0.0;

px[1][1] = 1.125; py[1][1] = -0;

px[1][2] = 0.625; py[1][2] = -0.75;

px[1][3] = -0.625; py[1][3] = -0.75;

px[1][4] = -0.375; py[1][4] = 0.0;

}

else {

//Set the coordinates of the body

px[0][0] = -1.125; py[0][0] = 0.0;

px[0][1] = -0.625; py[0][1] = 0.75;

px[0][2] = 0.625; py[0][2] = 0.75;

px[0][3] = 1.125; py[0][3] = -0;

px[0][4] = -1.125; py[0][4] = 0.0;

px[1][0] = -1.125; py[1][0] = 0.0;

px[1][1] = 1.125; py[1][1] = -0;

px[1][2] = 0.625; py[1][2] = -0.75;

px[1][3] = -0.625; py[1][3] = -0.75;

px[1][4] = -1.125; py[1][4] = 0.0;

}

//set the right foot

plx[0] = -0.25; ply[0] = -0.5;

plx[1] = -0.25; ply[1] = -1.0;

plx[2] = -0.50; ply[2] = -1.0;

plx[3] = -0.25; ply[3] = -1.0;

//set the left foot

pl2x[0] = 0.25; pl2y[0] = -0.5;

pl2x[1] = 0.25; pl2y[1] = -1.0;

pl2x[2] = 0.0; pl2y[2] = -1.0;

pl2x[3] = 0.25; pl2y[3] = -1.0;

//set the Closed mouth

pl3x[0] = -0.375; pl3y[0] = -0.0;

pl3x[1] = -0.875; pl3y[1] = -0.5;

pl3x[2] = -0.875; pl3y[2] = -0.5;

//set the eye

eye[0] = -0.5; eye[1] = 0.5;

pz[0] = 0.5;

pz[1] = -0.5;

return;

} //end of loadicon

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* function drawicon \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void drawicon(float px[][5], float py[][5], float pz[2], float plx[], float ply[], float pl2x[], float pl2y[], float pl3x[], float pl3y[], float eye[])

{

//draw the square icon at the transformed position

int i, face;

cout << "in drawicon" << endl;

glColor3f(1.0, 0.0, 0.0);

if (dx == 0) {

for (face = 0; face < 2; face++) {

for (int j = 0; j < 2; j++) {

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < 5; i++) {

glVertex3f(px[face][i], py[face][i], pz[j]);

}

glEnd();

}

}

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][0], py[1][0], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][0], py[1][0], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[0]);

glEnd();

}

else {

for (face = 0; face < 2; face++) {

for (int j = 0; j < 2; j++) {

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < 5; i++) {

glVertex3f(px[face][i], py[face][i], pz[j]);

}

glEnd();

}

}

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glEnd();

}

glColor3f(1.0, 1.0, 1.0);

//now draw the line

glBegin(GL\_LINES);

glVertex3f(plx[0], ply[0],pz[1]);

for (i = 1; i <= 3; i++)

{

glVertex3f(plx[i], ply[i],pz[1]);

}//glVertex2f(plxp[2],plyp[2]);

glEnd();

glBegin(GL\_LINES);

glVertex3f(pl2x[0], pl2y[0],pz[0]);

for (i = 1; i <= 3; i++)

{

glVertex3f(pl2x[i], pl2y[i],pz[0]);

}

glEnd();

//Draw Eye

glBegin(GL\_POINTS);

glColor3f(1.0, 0.0, 0.0);

glVertex3f(eye[0], eye[1],pz[0]);

glEnd();

glBegin(GL\_POINTS);

glColor3f(1.0, 0.0, 0.0);

glVertex3f(eye[0], eye[1],pz[1]);

//Draw mouth

glColor3f(0.0, 0.0, 0.0);

glEnd();

if (dx == 0) {//Open mouth

}

else {//ClosedMouth

glBegin(GL\_LINES);

glVertex3f(pl3x[1], pl3y[1],pz[0]);

glVertex3f(pl3x[0], pl3y[0],pz[0]);

glVertex3f(pl3x[2], pl3y[2],pz[0]);

glEnd();

glVertex3f(pl3x[1], pl3y[1],pz[1]);

glVertex3f(pl3x[0], pl3y[0],pz[1]);

glVertex3f(pl3x[2], pl3y[2],pz[1]);

}

glFlush();

return;

} //end of draw icon

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* function drawicon \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void drawicon2(float px[][5], float py[][5], float pz[2], float plx[], float ply[], float pl2x[], float pl2y[], float pl3x[], float pl3y[], float eye[])

{

//draw the square icon at the transformed position

int i, face;

cout << "in drawicon" << endl;

glColor3f(0.0, 1.0, 0.0);

if (dx == 0) {

for (face = 0; face < 2; face++) {

for (int j = 0; j < 2; j++) {

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < 5; i++) {

glVertex3f(px[face][i], py[face][i], pz[j]);

}

glEnd();

}

}

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][0], py[1][0], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][0], py[1][0], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[1][0], py[1][0], pz[0]);

glEnd();

}

else {

for (face = 0; face < 2; face++) {

for (int j = 0; j < 2; j++) {

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < 5; i++) {

glVertex3f(px[face][i], py[face][i], pz[j]);

}

glEnd();

}

}

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[1]);

glVertex3f(px[0][1], py[0][1], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[1]);

glVertex3f(px[0][2], py[0][2], pz[0]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[1]);

glVertex3f(px[0][3], py[0][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][2], py[1][2], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][2], py[1][2], pz[0]);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex3f(px[1][3], py[1][3], pz[0]);

glVertex3f(px[1][3], py[1][3], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[1]);

glVertex3f(px[0][0], py[0][0], pz[0]);

glVertex3f(px[1][3], py[1][3], pz[0]);

glEnd();

}

glColor3f(1.0, 1.0, 1.0);

//now draw the line

glBegin(GL\_LINES);

glVertex3f(plx[0], ply[0], pz[1]);

for (i = 1; i <= 3; i++)

{

glVertex3f(plx[i], ply[i], pz[1]);

}//glVertex2f(plxp[2],plyp[2]);

glEnd();

glBegin(GL\_LINES);

glVertex3f(pl2x[0], pl2y[0], pz[0]);

for (i = 1; i <= 3; i++)

{

glVertex3f(pl2x[i], pl2y[i], pz[0]);

}

glEnd();

//Draw Eye

glBegin(GL\_POINTS);

glColor3f(1.0, 0.0, 0.0);

glVertex3f(eye[0], eye[1], pz[0]);

glEnd();

glBegin(GL\_POINTS);

glColor3f(1.0, 0.0, 0.0);

glVertex3f(eye[0], eye[1], pz[1]);

//Draw mouth

glColor3f(0.0, 0.0, 0.0);

glEnd();

if (dy>-3.0) {//Open mouth

}

else {//ClosedMouth

glBegin(GL\_LINES);

glVertex3f(pl3x[1], pl3y[1], pz[0]);

glVertex3f(pl3x[0], pl3y[0], pz[0]);

glVertex3f(pl3x[2], pl3y[2], pz[0]);

glEnd();

glVertex3f(pl3x[1], pl3y[1], pz[1]);

glVertex3f(pl3x[0], pl3y[0], pz[1]);

glVertex3f(pl3x[2], pl3y[2], pz[1]);

}

glFlush();

return;

} //end of draw icon2

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* function settrans2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void settrans2(void)

/\*Sets the MODELVIEW MATRIX for the square. Note that the calls are done backqards

that is if we want to rotate and move the pattern, call glTranslate first and then glRotate \*/

{

cout << "in settrans2" << endl;

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(dx, dy, 0.0);

glRotatef(theta, 0.0, 0.0, 1.0);// note that the angle theta is in degrees, not radians

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* function settrans3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void settrans3(void)

/\*Sets the MODELVIEW MATRIX for the triangle. Note again that the calls are done backward.

Further note that we must have a MODELVIEW MATRIX for each figure \*/

{

cout << "in settrans3" << endl;

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(dxw, dyw, 0.0);

glRotatef(thetaw, zetaw, 0.0, 1.0);// note that the angle theta is in degrees, not radians

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Function SetupRC\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Setup the rendering state

void SetupRC(void)

{// this function sets the clear color of an open window and clears the open window

// Set clear color to blue

glClearColor(0.0, 0.0, 1.0, 1.0);

return;

}//end of SetupRC

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functioner Timer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TimerFunction(int value)

//this call back function is call each 30 ms and changes the location,scale and rotation

// of the square.

{

switch (frame)

{

case 1: //frame 1 polyman starts at right (7, -3) and walks to the middle right (3.5,-3)

//Polyman parameters

dx -= 0.15;

dxw += 0.05;

theta = theta - 5;

thetaw = thetaw - 2;

if (dx <= 0.0) {

dx = 0.0;

frame = 3;

}

else {

frame = 2;

}

break;

case 2: //frame 1 polyman starts at right (7, -3) and walks to the middle right (3.5,-3)

//Polyman parameters

theta = theta + 5;

thetaw = thetaw + 2;

if (dx <= 0.0) {

dx = 0.0;

frame = 3;

}

else {

frame = 1;

}

//we key on Polyman's position to change the frame

break;

case 3:// frame 2 polyman jumps to y=5

dy += 0.2;

if (dy > 5.0)

{

dy = 5.0;

frame = 4;

}

break;

case 4:// frame 3 Polyman rotates at x=3.5,y=5.0

theta += 5.0;

if (theta >= 360.0)

{

frame = 5;

theta = 0.0;

}

break;

case 5: // frame 4 Polyman moves down to x=0.0, y=-3.0

dy -= 0.2;

if (dy <= -3.0)

{

dy = -3.0;

frame = 6;

}

break;

case 6:

zetaw +zetaw +5.0;

if (zetaw <=0) {

frame = 8;

}

frame = 7;

break;

case 7:

zetaw = zetaw- 5.0;

if (zetaw == 0) {

frame = 7;

}

else {

frame = 8;

}

break;

case 8: //frame 1 polyman starts at right (7, -3) and walks to the middle right (3.5,-3)

//Polyman parameters

dx -= 0.15;

dxw -= 0.15;

thetaw = thetaw - 5;

theta = theta - 5;

if (dx <= -7) {

break;

}

else {

frame = 9;

}

break;

case 9: //frame 1 polyman starts at right (7, -3) and walks to the middle right (3.5,-3)

//Polyman parameters

theta = theta + 5;

thetaw = thetaw + 5;

if (dx <= -7) {

break;

}

else {

frame = 8;

}

//we key on Polyman's position to change the frame

break;

}

// Redraw the scene with new coordinates

glutPostRedisplay();

glutTimerFunc(100, TimerFunction, 1);

}

