1.write a program tp crate tringle reacange line loop unfilled polygon using opengl fcuntions

#include <GL/glut.h>

#include <cmath>

void init() {

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

}

void drawTriangle() {

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_TRIANGLES);

glVertex2f(-0.8f, -0.5f);

glVertex2f(-0.2f, -0.5f);

glVertex2f(-0.5f, 0.1f);

glEnd();

}

void drawRectangle() {

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_QUADS);

glVertex2f(0.2f, -0.5f);

glVertex2f(0.8f, -0.5f);

glVertex2f(0.8f, 0.1f);

glVertex2f(0.2f, 0.1f);

glEnd();

}

void drawHexagon() {

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

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 6; ++i) {

glVertex2f(0.3f \* cos(i \* 2.0f \* 3.1415926f / 6.0f),

0.3f \* sin(i \* 2.0f \* 3.1415926f / 6.0f));

}

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawTriangle();

drawRectangle();

glPushMatrix();

glTranslatef(0.0f, 0.6f, 0.0f);

drawHexagon();

glPopMatrix();

glFlush();

}

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

glutInit(&argc, argv);

glutCreateWindow("OpenGL Shapes with GLUT");

glutInitWindowSize(800, 600);

glutInitWindowPosition(50, 50);

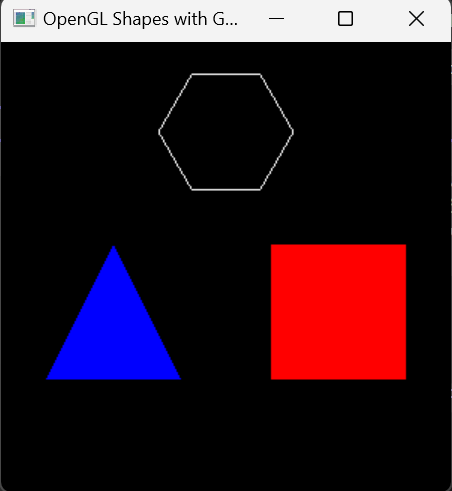
init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}



2.Write a program to create a hexagon then check heaxagons size using the window shape

#include <GL/glut.h>

#include <cmath>

#include <cstdlib>

const double TWO\_PI = 6.2831853;

GLsizei winwidth = 400, winheight = 400;

class ScreenPt {

public:

GLint x, y;

};

ScreenPt hexVertex[6];

GLint hexRadius = 120;

void updateHexVertices(GLsizei width, GLsizei height) {

ScreenPt center;

center.x = width / 2;

center.y = height / 2;

for (int k = 0; k < 6; ++k) {

double theta = TWO\_PI \* k / 6;

hexVertex[k].x = center.x + hexRadius \* cos(theta);

hexVertex[k].y = center.y + hexRadius \* sin(theta);

}

}

void init() {

glClearColor(1.0, 0.0, 1.0, 0.0); // Set the background color to purple

glColor3f(1.0, 1.0, 0.0); // Set the drawing color to yellow

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, winwidth, 0.0, winheight);

updateHexVertices(winwidth, winheight);

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POLYGON);

for (int k = 0; k < 6; ++k) {

glVertex2i(hexVertex[k].x, hexVertex[k].y);

}

glEnd();

glFlush();

}

void reshape(int newWidth, int newHeight) {

glViewport(0, 0, (GLsizei)newWidth, (GLsizei)newHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, (GLdouble)newWidth, 0.0, (GLdouble)newHeight);

updateHexVertices(newWidth, newHeight);

display();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(winwidth, winheight);

glutCreateWindow("Hexagon Reshape Example");

init();

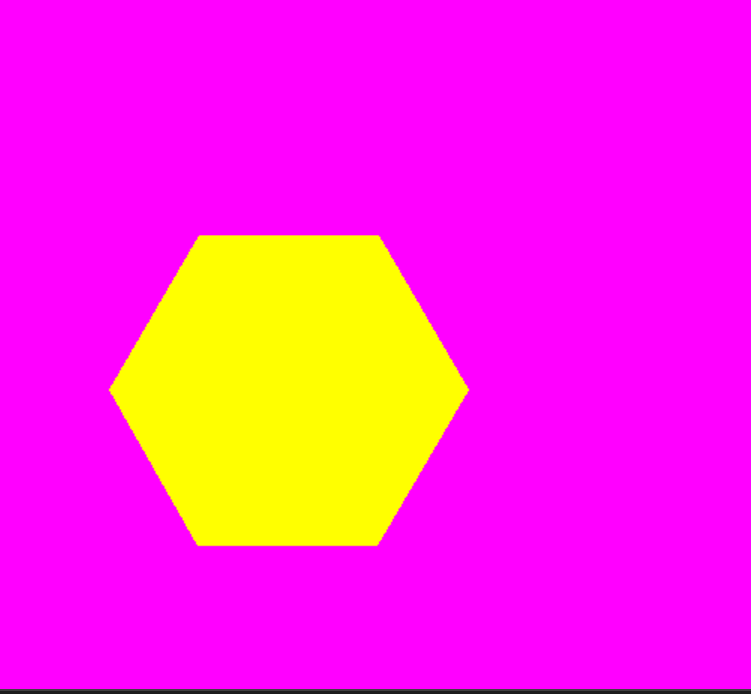
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



3.Program to perform gyrometric operations for 2d objects (resize reshape rotate and movement of an object from one to palce another)



#include <GL/glut.h>

#include <cmath>

float angle = 0.0f; // Rotation angle

float scaleX = 1.0f; // Scale factor along the X-axis

float scaleY = 1.0f; // Scale factor along the Y-axis

float transX = 0.0f; // Translation along the X-axis

float transY = 0.0f; // Translation along the Y-axis

int startX, startY; // Initial mouse click position

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white

glColor3f(0.0, 0.0, 0.0); // Set drawing color to black

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 800.0, 0.0, 600.0); // Set coordinate system

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix(); // Save the current transformation matrix

glTranslatef(transX, transY, 0.0f); // Apply translation

glRotatef(angle, 0.0f, 0.0f, 1.0f); // Apply rotation

glScalef(scaleX, scaleY, 1.0f); // Apply scaling

// Draw a rectangle

glBegin(GL\_POLYGON);

glVertex2f(-50.0f, -25.0f);

glVertex2f(50.0f, -25.0f);

glVertex2f(50.0f, 25.0f);

glVertex2f(-50.0f, 25.0f);

glEnd();

glPopMatrix(); // Restore the previous transformation matrix

glFlush();

}

void reshape(int w, int h) {

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);

}

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

switch (key) {

case 'r': angle += 10.0f; break; // Rotate clockwise

case 'R': angle -= 10.0f; break; // Rotate counterclockwise

case 'x': scaleX += 0.1f; break; // Scale up along X-axis

case 'X': scaleX -= 0.1f; break; // Scale down along X-axis

case 'y': scaleY += 0.1f; break; // Scale up along Y-axis

case 'Y': scaleY -= 0.1f; break; // Scale down along Y-axis

case 27: exit(0); // Escape key to exit

}

glutPostRedisplay();

}

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

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

startX = x;

startY = glutGet(GLUT\_WINDOW\_HEIGHT) - y;

}

}

void motion(int x, int y) {

int endX = x;

int endY = glutGet(GLUT\_WINDOW\_HEIGHT) - y;

transX += (endX - startX);

transY += (endY - startY);

startX = endX;

startY = endY;

glutPostRedisplay();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("2D Geometric Transformations");

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMouseFunc(mouse);

glutMotionFunc(motion);

glutMainLoop();

return 0;

}

4.Program to perform gyrometric operations for 3d objects (resize reshape rotate and movement of an object from one to palce another)

#include <GL/glut.h>

#include <iostream>

float angle = 0.0f;

float scale = 1.0f;

float tx = 0.0f, ty = 0.0f, tz = 0.0f;

void drawCube() {

glBegin(GL\_QUADS);

// Front face

glColor3f(1.0f, 0.0f, 0.0f); // Red

glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f, 1.0f);

glVertex3f(1.0f, -1.0f, 1.0f);

// Back face

glColor3f(0.0f, 1.0f, 0.0f); // Green

glVertex3f(1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, -1.0f);

// Left face

glColor3f(0.0f, 0.0f, 1.0f); // Blue

glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(-1.0f, -1.0f, 1.0f);

// Right face

glColor3f(1.0f, 1.0f, 0.0f); // Yellow

glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, 1.0f);

// Top face

glColor3f(1.0f, 0.0f, 1.0f); // Magenta

glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(1.0f, 1.0f, -1.0f);

// Bottom face

glColor3f(0.0f, 1.0f, 1.0f); // Cyan

glVertex3f(1.0f, -1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, -1.0f);

glEnd();

}

void display() {

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

glLoadIdentity();

glTranslatef(tx, ty, tz - 5.0f); // Ensure the object is within the view frustum

glScalef(scale, scale, scale);

glRotatef(angle, 1.0f, 1.0f, 1.0f);

drawCube();

glutSwapBuffers();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0f, (GLfloat)w / (GLfloat)h, 0.1f, 100.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

switch (key) {

case 'w': ty += 0.1f; break;

case 's': ty -= 0.1f; break;

case 'a': tx -= 0.1f; break;

case 'd': tx += 0.1f; break;

case '+': scale += 0.1f; break;

case '-': scale -= 0.1f; break;

case 'r': angle += 5.0f; break;

case 27: exit(0); break; // ESC key to exit

}

glutPostRedisplay();

}

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("3D Transformations");

glEnable(GL\_DEPTH\_TEST);

glutDisplayFunc(display);

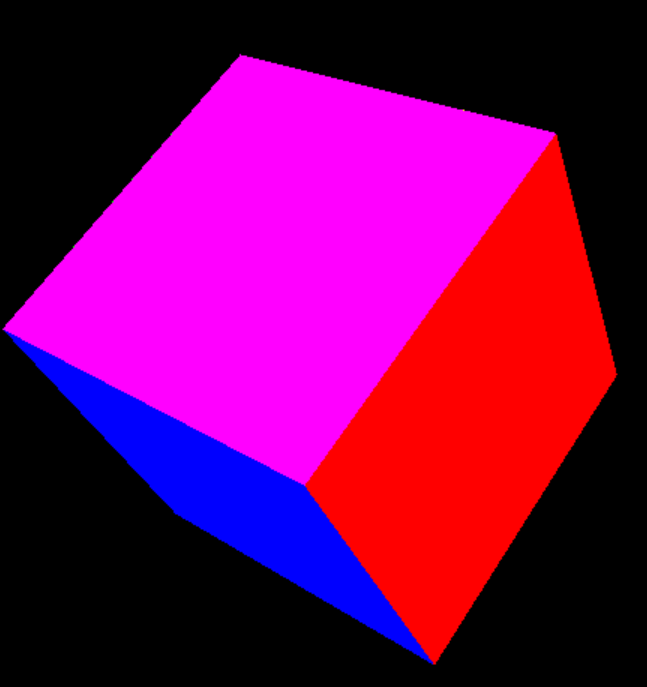
glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

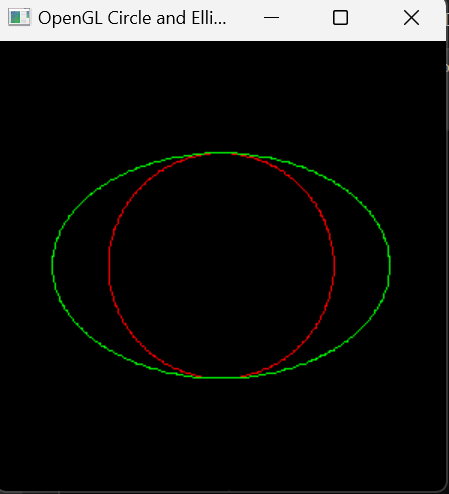
glutMainLoop();

return 0;

}



5.program to draw circlwa and ecclipses using algo



#include <GL/glut.h>

#include <cmath>

// Function to set a pixel in the window

void setPixel(int x, int y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

// Midpoint Circle Algorithm

void drawCircle(int xc, int yc, int r) {

int x = 0;

int y = r;

int d = 1 - r;

setPixel(xc + x, yc + y);

setPixel(xc - x, yc + y);

setPixel(xc + x, yc - y);

setPixel(xc - x, yc - y);

setPixel(xc + y, yc + x);

setPixel(xc - y, yc + x);

setPixel(xc + y, yc - x);

setPixel(xc - y, yc - x);

while (x < y) {

if (d < 0) {

d += 2 \* x + 3;

}

else {

d += 2 \* (x - y) + 5;

y--;

}

x++;

setPixel(xc + x, yc + y);

setPixel(xc - x, yc + y);

setPixel(xc + x, yc - y);

setPixel(xc - x, yc - y);

setPixel(xc + y, yc + x);

setPixel(xc - y, yc + x);

setPixel(xc + y, yc - x);

setPixel(xc - y, yc - x);

}

}

// Midpoint Ellipse Algorithm

void drawEllipse(int xc, int yc, int rx, int ry) {

float dx, dy, d1, d2, x, y;

x = 0;

y = ry;

// Initial decision parameter of region 1

d1 = (ry \* ry) - (rx \* rx \* ry) + (0.25 \* rx \* rx);

dx = 2 \* ry \* ry \* x;

dy = 2 \* rx \* rx \* y;

// For region 1

while (dx < dy) {

setPixel(xc + x, yc + y);

setPixel(xc - x, yc + y);

setPixel(xc + x, yc - y);

setPixel(xc - x, yc - y);

if (d1 < 0) {

x++;

dx = dx + (2 \* ry \* ry);

d1 = d1 + dx + (ry \* ry);

}

else {

x++;

y--;

dx = dx + (2 \* ry \* ry);

dy = dy - (2 \* rx \* rx);

d1 = d1 + dx - dy + (ry \* ry);

}

}

// Decision parameter of region 2

d2 = ((ry \* ry) \* ((x + 0.5) \* (x + 0.5))) + ((rx \* rx) \* ((y - 1) \* (y - 1))) - (rx \* rx \* ry \* ry);

// For region 2

while (y >= 0) {

setPixel(xc + x, yc + y);

setPixel(xc - x, yc + y);

setPixel(xc + x, yc - y);

setPixel(xc - x, yc - y);

if (d2 > 0) {

y--;

dy = dy - (2 \* rx \* rx);

d2 = d2 + (rx \* rx) - dy;

}

else {

y--;

x++;

dx = dx + (2 \* ry \* ry);

dy = dy - (2 \* rx \* rx);

d2 = d2 + dx - dy + (rx \* rx);

}

}

}

// Display callback function

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0f, 0.0f, 0.0f); // Red color

drawCircle(0, 0, 100); // Draw a circle with radius 100

glColor3f(0.0f, 1.0f, 0.0f); // Green color

drawEllipse(0, 0, 150, 100); // Draw an ellipse with radii 150 and 100

glFlush();

}

void init() {

glClearColor(0.0, 0.0, 0.0, 0.0); // Set the background color to black

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-200, 200, -200, 200); // Set the coordinate system

}

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

glutInit(&argc, argv);

glutCreateWindow("OpenGL Circle and Ellipse");

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

6. program to change colour of a triangle by integrating mouse functions

#include <GL/glut.h>

#include <cstdlib> // For rand() and srand()

#include <ctime> // For time()

// Global variables to store the color of the triangle

float red = 1.0f, green = 0.0f, blue = 0.0f;

// Function to draw a triangle

void drawTriangle() {

glBegin(GL\_TRIANGLES);

glColor3f(red, green, blue); // Set the current color

glVertex2f(-0.5f, -0.5f);

glVertex2f(0.5f, -0.5f);

glVertex2f(0.0f, 0.5f);

glEnd();

}

// Display callback function

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawTriangle();

glFlush();

}

// Mouse callback function

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

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

// Generate random colors

red = static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX);

green = static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX);

blue = static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX);

// Request a redisplay

glutPostRedisplay();

}

}

void init() {

glClearColor(0.0, 0.0, 0.0, 0.0); // Set the background color to black

}

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

srand(static\_cast<unsigned>(time(0))); // Seed the random number generator

glutInit(&argc, argv);

glutCreateWindow("OpenGL Triangle Color Change");

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

init();

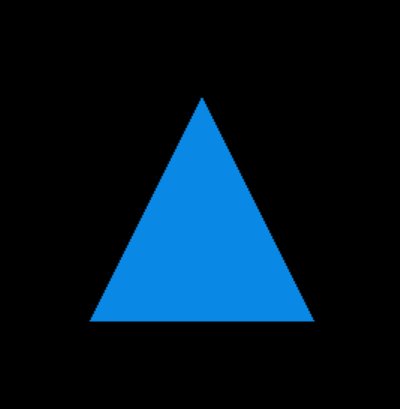
glutDisplayFunc(display);

glutMouseFunc(mouse);

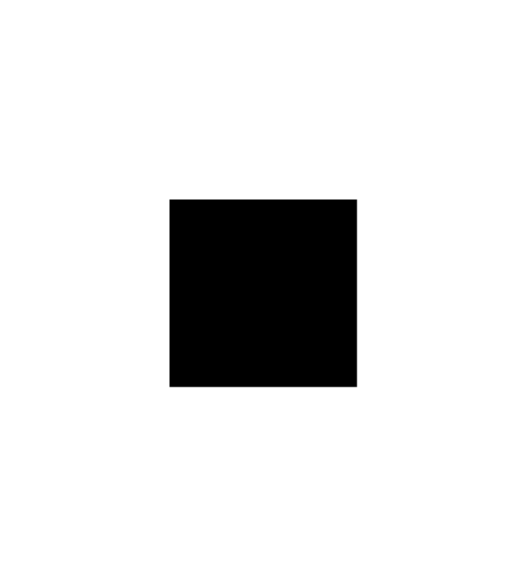
glutMainLoop();

return 0;

}



7.PROGRAM TO INC AND DEC THE SIZE OF SQUARE USING MOUSE AND KEYBORAD FUNCTION



#include <GL/glut.h>

// Initial size of the square

float size = 0.5f;

// Function to draw a square

void drawSquare(float centerX, float centerY, float size) {

float halfSize = size / 2.0f;

glBegin(GL\_QUADS);

glVertex2f(centerX - halfSize, centerY - halfSize);

glVertex2f(centerX + halfSize, centerY - halfSize);

glVertex2f(centerX + halfSize, centerY + halfSize);

glVertex2f(centerX - halfSize, centerY + halfSize);

glEnd();

}

// Display callback function

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawSquare(0.0f, 0.0f, size);

glFlush();

}

// Keyboard callback function

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

if (key == 'i' || key == 'I') {

size += 0.1f; // Increase size

}

else if (key == 'd' || key == 'D') {

size -= 0.1f; // Decrease size

}

glutPostRedisplay(); // Request a redraw

}

// Mouse callback function

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

if (state == GLUT\_DOWN) {

if (button == GLUT\_LEFT\_BUTTON) {

size += 0.1f; // Increase size

}

else if (button == GLUT\_RIGHT\_BUTTON) {

size -= 0.1f; // Decrease size

}

glutPostRedisplay(); // Request a redraw

}

}

// Initialization function

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0); // White background

glColor3f(0.0, 0.0, 0.0); // Black drawing color

gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // Define the viewing area

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("Resize Square");

init();

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

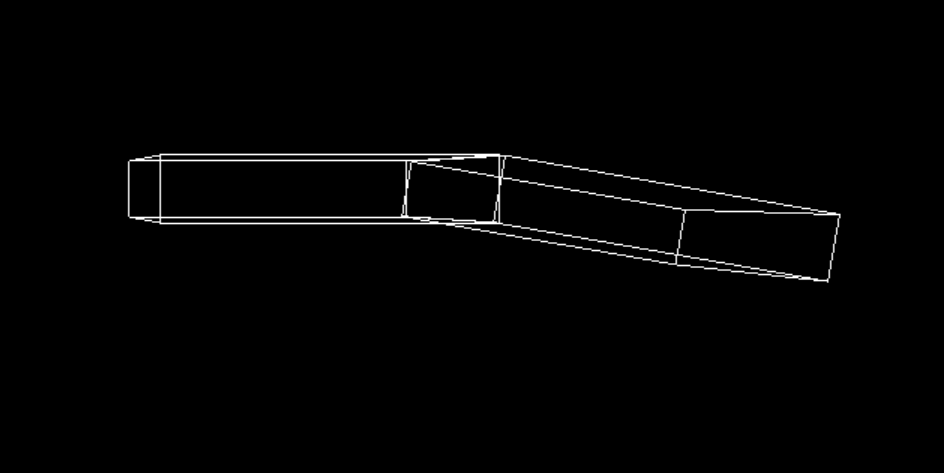
glutMouseFunc(mouse);

glutMainLoop();

return 0;

}

8. A PROGRAM TO PERFORM ROBOTIC ARM ROATION KEYBORAD FUNCTION



#include<GL/glut.h>

static int shoulder = 0, elbow = 0;

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_FLAT);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glTranslatef(1.0, 0.0, 0.0);

glRotatef((GLfloat)shoulder, 0.0, 0.0, 1.0);

glTranslatef(1.0, 0.0, 0.0);

glPushMatrix();

glScalef(2.0, 0.4, 1.0);

glutWireCube(1.0);

glPopMatrix();

glTranslatef(1.0, 0.0, 0.0);

glRotatef((GLfloat)elbow, 0.0, 0.0, 1.0);

glTranslatef(1.0, 0.0, 0.0);

glPushMatrix();

glScalef(2.0, 0.4, 1.0);

glutWireCube(1.0);

glPopMatrix();

glPopMatrix();

glutSwapBuffers();

}

void reshape(int w, int h)

{

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

}

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

{

switch (key)

{

case's':shoulder = (shoulder + 5) % 360;

glutPostRedisplay();

break;

case'S':shoulder = (shoulder - 5) % 360;

glutPostRedisplay();

break;

case'e':elbow = (elbow + 5) % 360;

glutPostRedisplay();

break;

case'E':elbow = (elbow - 5) % 360;

glutPostRedisplay();

break;

default:break;

}

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow(argv[0]);

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

9.A PROGRAMT TO 3D OBJECT ANDPERFORM DIFFENT VIEWS (PROSPECTIVE , ORTHOHGRAPHIC, OBLIQUE)

#include <GL/glut.h>

#include <stdlib.h>

// Global variables

int viewMode = 0; // 0: Perspective, 1: Orthographic, 2: Oblique

void init() {

glClearColor(0.0, 0.0, 0.0, 1.0);

glEnable(GL\_DEPTH\_TEST);

}

void setView() {

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (viewMode == 0) {

// Perspective view

gluPerspective(45.0, 1.0, 1.0, 10.0);

}

else if (viewMode == 1) {

// Orthographic view

glOrtho(-2.0, 2.0, -2.0, 2.0, 1.0, 10.0);

}

else if (viewMode == 2) {

// Oblique view

glLoadIdentity(); // Reset the modelview matrix

// Apply transformations directly

glTranslatef(0.0, 0.0, -2.0);

glRotatef(45.0, 0.0, 1.0, 0.0);

// Set up the orthographic projection separately

glOrtho(-2.0, 2.0, -2.0, 2.0, 1.0, 10.0);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

}

void drawCube() {

glBegin(GL\_QUADS);

// Front face

glColor3f(1.0, 0.0, 0.0); // Red

glVertex3f(-1.0, -1.0, 1.0);

glVertex3f(1.0, -1.0, 1.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(-1.0, 1.0, 1.0);

// Back face

glColor3f(0.0, 1.0, 0.0); // Green

glVertex3f(-1.0, -1.0, -1.0);

glVertex3f(1.0, -1.0, -1.0);

glVertex3f(1.0, 1.0, -1.0);

glVertex3f(-1.0, 1.0, -1.0);

// Top face

glColor3f(0.0, 0.0, 1.0); // Blue

glVertex3f(-1.0, 1.0, -1.0);

glVertex3f(-1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, -1.0);

// Bottom face

glColor3f(1.0, 1.0, 0.0); // Yellow

glVertex3f(-1.0, -1.0, -1.0);

glVertex3f(1.0, -1.0, -1.0);

glVertex3f(1.0, -1.0, 1.0);

glVertex3f(-1.0, -1.0, 1.0);

// Right face

glColor3f(1.0, 0.0, 1.0); // Magenta

glVertex3f(1.0, -1.0, -1.0);

glVertex3f(1.0, 1.0, -1.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, -1.0, 1.0);

// Left face

glColor3f(0.0, 1.0, 1.0); // Cyan

glVertex3f(-1.0, -1.0, -1.0);

glVertex3f(-1.0, -1.0, 1.0);

glVertex3f(-1.0, 1.0, 1.0);

glVertex3f(-1.0, 1.0, -1.0);

glEnd();

}

void display() {

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

setView();

drawCube();

glutSwapBuffers();

}

void reshape(int width, int height) {

glViewport(0, 0, width, height);

setView();

}

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

switch (key) {

case '1':

viewMode = 0; // Perspective view

break;

case '2':

viewMode = 1; // Orthographic view

break;

case '3':

viewMode = 2; // Oblique view

break;

case 27: // ESC key

exit(0);

break;

}

glutPostRedisplay();

}

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("3D Object Viewer");

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

10.pogram to crate in 3d using opengl

#include <GL/glut.h>

void init() {

glClearColor(0.7, 0.9, 1.0, 1.0); // Light blue background color

glEnable(GL\_DEPTH\_TEST); // Enable depth testing for 3D

}

void drawGround() {

glBegin(GL\_QUADS);

glColor3f(0.6, 0.8, 0.2); // Green ground color

glVertex3f(-5.0, 0.0, -5.0);

glVertex3f(-5.0, 0.0, 5.0);

glVertex3f(5.0, 0.0, 5.0);

glVertex3f(5.0, 0.0, -5.0);

glEnd();

}

void drawTree() {

glColor3f(0.8, 0.5, 0.2); // Brown trunk color

glPushMatrix();

glTranslatef(-1.5, 0.0, -1.5);

glScalef(0.3, 1.5, 0.3);

glutSolidCube(1.0); // Trunk

glColor3f(0.0, 0.6, 0.1); // Green leaf color

glTranslatef(0.0, 0.8, 0.0);

glutSolidSphere(0.8, 10, 10); // Leaves

glPopMatrix();

}

void drawSun() {

glColor3f(1.0, 1.0, 0.0); // Yellow sun color

glPushMatrix();

glTranslatef(3.0, 3.0, -3.0);

glutSolidSphere(1.0, 20, 20); // Sun

glPopMatrix();

}

void display() {

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

glLoadIdentity();

// Set up the camera

gluLookAt(3.0, 3.0, 5.0, // eye position

0.0, 0.0, 0.0, // look-at position

0.0, 1.0, 0.0); // up direction

drawGround();

drawTree();

glPushMatrix();

glTranslatef(2.0, 0.0, 2.0);

drawTree();

glPopMatrix();

drawSun();

glutSwapBuffers();

}

void reshape(int width, int height) {

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, (double)width / height, 1.0, 20.0); // perspective parameters

glMatrixMode(GL\_MODELVIEW);

}

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("3D Scenery");

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}