

| **TITLE**: WAP to implement Simple Interaction with the mouse and keyboard. |
| --- |

**AIM:**

Write an OpenGL program to demonstrate use of interaction through mouse and keyboard.

(Example: Pressing ‘p’ draws a dot at the mouse position; pressing the left arrow key adds a point to some (global) list, but does no drawing; pressing ‘E’ exits from the program.)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

CO1: Understand the basic concepts of computer graphics and OpenGL

CO4: Understand the computer Input& interaction, Curves and Computer Animation

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

* GeeksforGeeks
* Google

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm/ Pseudocode for each process:**

**1.** **Keyboard Input Handling**

**Define a function to handle keyboard inputs:**

* + **If 'p' is pressed:**
    - **Add a point at the current mouse position to points.**
    - **Trigger a redraw.**
  + **If 'r' is pressed:**
    - **Set randomColor to true and generate random RGB values for the point color.**
    - **Trigger a redraw.**
  + **If 'c' is pressed:**
    - **Increase pointSize by 2.**
    - **Reset to default if pointSize exceeds a maximum value.**
    - **Trigger a redraw.**
  + **If 'E' is pressed:**
    - **Exit the program.**

**2.** **Special Key Handling**

**Define a function to handle special key inputs:**

* + **If the left arrow key is pressed:**
    - **Store the current mouse position in points.**
    - **Print the stored position to the console.**
  + **If the right arrow key is pressed:**
    - **Trigger a redraw.**

**3.**  **Mouse Input Handling**

**Define a function to handle mouse button events:**

* + **If the right mouse button is clicked:**
    - **Clear all points in points.**
    - **Print a message to the console.**
    - **Trigger a redraw.**

**4.** **Mouse Motion Handling**

**Define a function to update mouseX and mouseY with the current mouse position.**

**5.** **Main Function**

* **Call the initialization functions.**
* **Register callback functions for display, keyboard, special keys, mouse, and mouse motion.**
* **Enter the main event loop.**

**Implementation details:**

#include <GL/glut.h>

#include <vector>

#include <iostream>

#include <cstdlib>

#include <ctime>

// Global variables

std::vector<std::pair<int, int>> points;

int mouseX, mouseY;

bool randomColor = false;

float pointSize = 5.0f;

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

// Function to display content on the screen

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Set point size

glPointSize(pointSize);

// Set color

if (randomColor) {

glColor3f(red, green, blue); // Random color when 'r' is pressed

} else {

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

}

glBegin(GL\_POINTS);

for (const auto& point : points) {

glVertex2i(point.first, point.second);

}

glEnd();

glutSwapBuffers();

}

// Function to handle keyboard input

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

switch (key) {

case 'p': // Press 'p' to draw a point at the mouse position

points.emplace\_back(mouseX, 600 - mouseY);

glutPostRedisplay(); // Trigger re-drawing

break;

case 'r': // Press 'r' for random color mode

randomColor = true;

// Generate random color

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

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

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

glutPostRedisplay();

break;

case 'c': // Press 'c' to change point size

pointSize += 2.0f;

if (pointSize > 20.0f) pointSize = 5.0f;

glutPostRedisplay();

break;

case 'E': // Press 'E' to exit

exit(0);

break;

default:

break;

}

}

// Function to handle special keys (like arrow keys)

void specialKeys(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_LEFT:

points.emplace\_back(mouseX, 600 - mouseY);

std::cout << "Stored Point: (" << mouseX << ", " << 600 - mouseY << ")" << std::endl;

break;

case GLUT\_KEY\_RIGHT:

glutPostRedisplay();

break;

default:

break;

}

}

// Function to handle mouse input

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

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

points.clear(); // Clear all points when right-click is detected

std::cout << "Screen cleared!" << std::endl;

glutPostRedisplay();

}

}

// Function to handle mouse motion

void mouseMove(int x, int y) {

mouseX = x;

mouseY = y;

}

// Initialization function

void init() {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 800, 0, 600);

srand(static\_cast<unsigned int>(time(0)));

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutCreateWindow("Enhanced OpenGL Mouse and Keyboard Interaction");

init();

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutSpecialFunc(specialKeys);

glutMouseFunc(mouse);

glutPassiveMotionFunc(mouseMove);

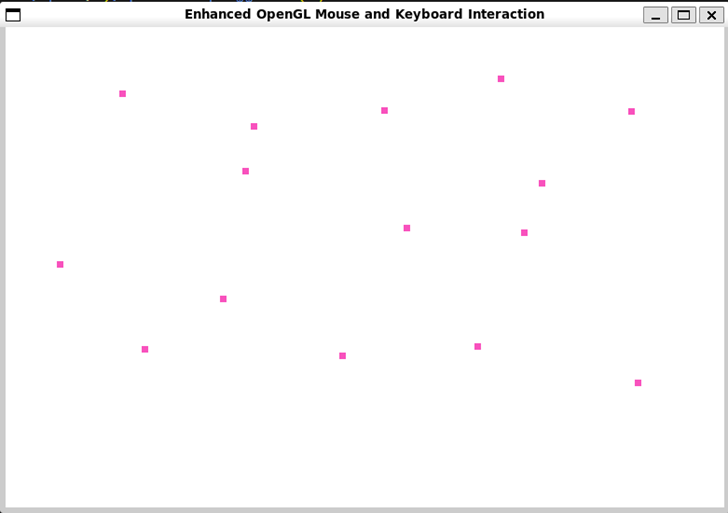
glutMainLoop();

return 0;

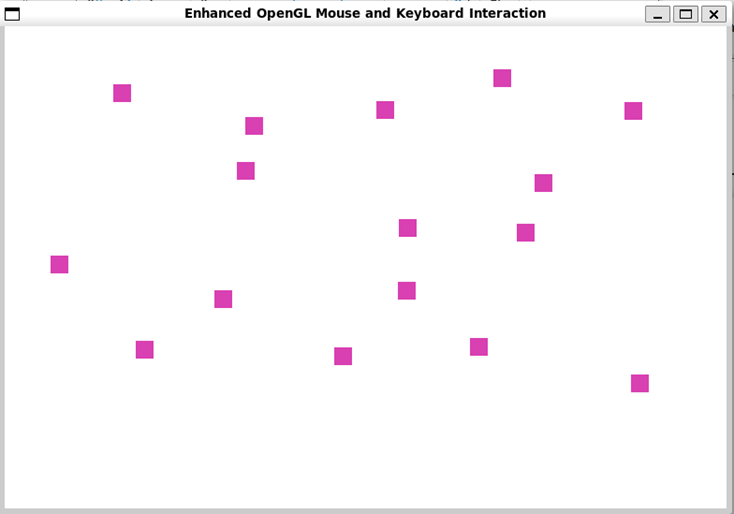
}

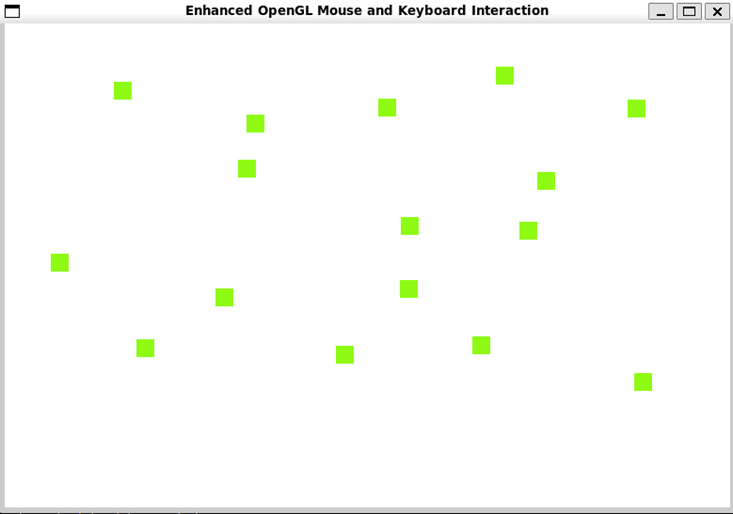
**Output(s) (Screen Shot):**

**1.** **Pressing 'p' draws a dot at the mouse position.**

****

**2.** **Pressing 'c' to change the size of the points.**

****

**3.** **Pressing 'r' to draw random-colored points.   
 **

**4.** **Right-click to clear the screen.  
**

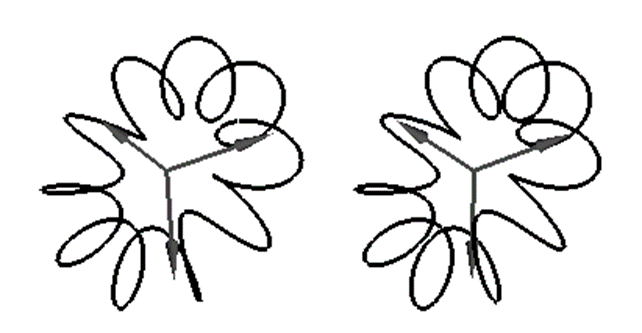
**5.** **Pressing 'E' exits the program.**

**Conclusion and discussion:**

**The OpenGL program developed demonstrates fundamental principles of interactive graphics programming using mouse and keyboard inputs.**

**Date: 07/10/2024 Signature of faculty in-charge**

**Post lab Question**

**Write a program to draw the following  
**

#include <GL/glut.h>

#include <cmath>

const int POINTS = 1000; // Number of points on the flower curve

const float PI = 3.14159265358979323846f;

// Function to draw the parametric flower curve

void drawFlowerCurve() {

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

glLineWidth(2.0f);

// Start drawing the curve

glBegin(GL\_LINE\_LOOP);

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

float t = (2 \* PI \* i) / POINTS; // Parametric angle

float x = (0.3 \* cos(3 \* t)) \* cos(t); // Parametric equation for x

float y = (0.3 \* cos(3 \* t)) \* sin(t); // Parametric equation for y

float z = 0.1 \* sin(2 \* t); // Parametric equation for z

glVertex3f(x, y, z);

}

glEnd();

}

// Display function for rendering

void display() {

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

glLoadIdentity();

gluLookAt(0.0, 0.0, 2.0, // Camera position

0.0, 0.0, 0.0, // Look at the origin

0.0, 1.0, 0.0); // Up vector

// Draw flower curve

drawFlowerCurve();

glutSwapBuffers();

}

// Initialization function

void init() {

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

glEnable(GL\_DEPTH\_TEST);

}

// Main function

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("Flower-like Curve");

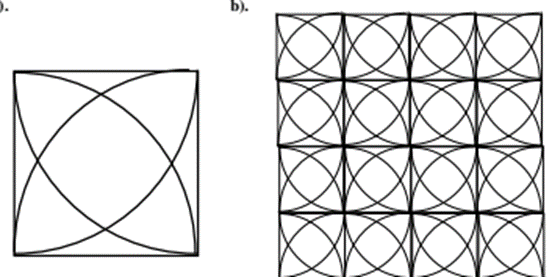
init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}



#include <GL/glut.h>

#include <cmath> // For math functions like sin, cos

const int grid\_size = 4; // Grid of 4x4

// Function to draw an arc between two points in a square

void drawArc(float cx, float cy, float r, int start\_angle, int end\_angle) {

glBegin(GL\_LINE\_STRIP);

for (int angle = start\_angle; angle <= end\_angle; angle++) {

float theta = angle \* 3.1415926 / 180;

float x = r \* cos(theta

float y = r \* sin(theta);

glVertex2f(x + cx, y + cy);

}

glEnd();

}

// Function to draw the grid with arcs in each square

void drawGridWithArcs() {

float size = 0.4f;

float offset = size / 2;

glColor3f(0.0f, 0.0f, 0.0f); // Black color for arcs

glLineWidth(2.0f);

// Loop through the grid

for (int i = -grid\_size / 2; i < grid\_size / 2; ++i) {

for (int j = -grid\_size / 2; j < grid\_size / 2; ++j) {

float x = i \* size; // x-coordinate for the square's center

float y = j \* size; // y-coordinate for the square's center

// Draw the square (grid cell)

glBegin(GL\_LINE\_LOOP);

glVertex2f(x - offset, y - offset); // Bottom-left corner

glVertex2f(x + offset, y - offset); // Bottom-right corner

glVertex2f(x + offset, y + offset); // Top-right corner

glVertex2f(x - offset, y + offset); // Top-left corner

glEnd();

// Draw arcs connecting opposite corners (quarter circles)

// Bottom-left corner to top-right corner

drawArc(x - offset, y - offset, size, 0, 90);

// Bottom-right corner to top-left corner

drawArc(x + offset, y - offset, size, 90, 180);

// Top-right corner to bottom-left corner

drawArc(x + offset, y + offset, size, 180, 270);

// Top-left corner to bottom-right corner

drawArc(x - offset, y + offset, size, 270, 360);

}

}

}

// Display function for rendering

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

// Draw the grid with arcs

drawGridWithArcs();

glutSwapBuffers();

}

// Initialization function

void init() {

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

}

// Main function

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(800, 800);

glutCreateWindow("Grid with Quarter-Circle Arcs");

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}