Lab04 for CPT205 Computer Graphics

Part 1 – OpenGL Exercise: Interactions using Mouse and Keyboard

The following sample program demonstrates the use of mouse and keyboard for interactions between the user and the program. It also shows how an object can be moved continuously or animated.

1) **Task 1**: Type in the sample code and run it to see what happens.

```
// File ID: Lab04-1.cpp
// Title: Interactions using Mouse and Keyboard / Animation
// Author:

#define FREEGLUT_STATIC
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <GL/freeglut.h>

typedef struct { GLfloat x, y; } point;  // define a 2D point
point p0 = {0,50};  // set initial co-ordinate values of the point
GLfloat finishx = 300;  // declare position of vertical axis

//Task 2
//Task 3
//Task 4

void display(void)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluOrtho2D(0, 600, 0, 400);

    glClearColor(0, 0, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 1, 1);
    glBegin(GL_LINES);
    glVertex2f(finishx, 50);
    glVertex2f(finishx, 350);
    glEnd();

    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2f(0, 0);
    glVertex2f(0, 50);
    glVertex2f(600, 50);
    glVertex2f(600, 0);
    glEnd();

    glColor3f(1, 0, 0);
    glBegin(GL_POLYGON);
    glVertex2f(p0.x, p0.y);
    glVertex2f(p0.x, p0.y + 20);
    glVertex2f(p0.x + 30, p0.y + 20);
    glVertex2f(p0.x + 30, p0.y);
    glEnd();

    glutSwapBuffers();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(600, 400);
    glutCreateWindow("My Interactive Window");

    glutDisplayFunc(display);
    //Task 2
    //Task 3
    //Task 4

    glutMainLoop();
}
```

2) **Task 2**: Add a time callback function to update parameter with a pre-set gap.

Add the following codes before the main() function:

```
GLfloat step = 10;  // declare incremental step
int time_interval = 16;  // declare refresh interval in ms

void when_in_mainloop()
{  // idle callback function
    glutPostRedisplay();  // force OpenGL to redraw the current window
}

void OnTimer(int value)
{
    p0.x += step;
    if (p0.x >= 600)
        p0.x = 0;
    else if (p0.x <= 0)
        p0.x = 599;

    glutTimerFunc(time_interval, OnTimer, 1);
}
```

Add the following codes in the main() function:

```
glutIdleFunc(when_in_mainloop);

/* glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background
processing tasks or continuous animation when window system events are not being received.

If enabled, the idle callback is continuously called when events are not being received. The callback
routine has no parameters. The current window and current menu will not be changed before the idle
callback. Programs with multiple windows and/or menus should explicitly set the current window and/or
current menu and not rely on its current setting. The amount of computation and rendering done in an idle
callback should be minimized to avoid affecting the program's interactive response. In general, not more
than a single frame of rendering should be done in an idle callback.

Assigning NULL to glutIdleFunc disables the generation of the idle callback. */

glutTimerFunc(time_interval, OnTimer, 1);
```

3) **Task 3**: Add a keyboard callback function to receive keyboard input

Add the following codes before the main() function:

```
void keyboard_input(unsigned char key, int x, int y) // keyboard interactions
{
    if (key == 'q' || key == 'Q')
        exit(0);
    else if (key == 'f' || key == 'F') // change direction of movement
        step = -step;
    else if (key == 's' || key == 'S') // stop movement
        step = 0;
    else if (key == 'r' || key == 'R') // reset step (resume movement)
        step = 10;
}
```

Add the following codes in the main() function:

```
glutKeyboardFunc(keyboard_input);  // keyboard callback function
```

4) **Task 4**: Add a mouse callback function to receive mouse input

Add the following codes before the main() function:

```
void mouse_input(int button, int state, int x, int y) // mouse interactions
{
    if (state == GLUT_DOWN && button == GLUT_LEFT_BUTTON && step >= -15)
        step -= 1;  // decrement step
    else if (state == GLUT_DOWN && button == GLUT_RIGHT_BUTTON && step <= 15)
        step += 1;  // increment step
}
```

Add the following codes in the main() function:

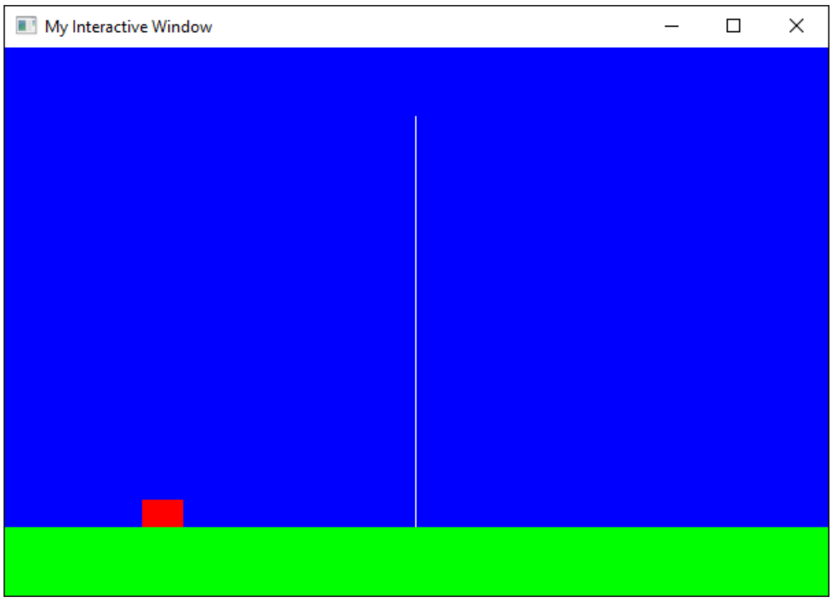
```
glutMouseFunc(mouse_input); // mouse callback function
```

5) **Task 5**: Try to modify some of the parameters to see the effect.

6) **Task 6**: Try to use different functions to achieve further features.

7) **Task 7**: Try to see if you could apply geometric transformations in the code and achieve similar effect.

8) **Task 8**: You should keep in mind how the functions work and can be called. You may now want to revisit the tasks and functions used.



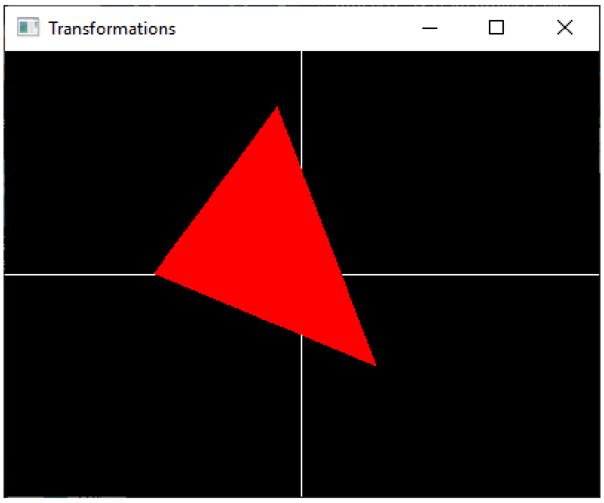
Part 2a - Geometric Transformations / Creation of Geometry in OpenGL

1) **Task 1**: First carefully read and understand the sample code below and then run it to see what happens. You should see a window shown in the next page.

```cpp
// File ID: Lab04-2a.cpp
// Title: Geometric Transformations
// Author:

#define FREEGLUT_STATIC
#include <math.h>
#include <GL/freeglut.h>
#include <iostream>

int width = 400;  // window with
int height = 300;  // window height

float t = 0.0;  // increment for translation
float r = 0.0;  // increment for rotation
float s = 1.0;  // increment for scaling

void when_in_mainloop()  // idle callback function
{
    glutPostRedisplay();  // force OpenGL to redraw the current window
}

void keyboard_input(unsigned char key, int x, int y)  // keyboard interaction
{
    if (key == 'q' || key == 'Q')
        exit(0);
    else if (key == 't' || key == 'T')
        t += 0.05;
    else if (key == 'b' || key == 'B')
        t -= 0.05;
    else if (key == 'r' || key == 'R')
        r--;
    else if (key == 'a' || key == 'A')
        r++;
    else if (key == 'i' || key == 'I')
        s += 0.01;
    else if (key == 'd' || key == 'D')
        s -= 0.01;
}

void DrawTriangles()
{
    glColor3f(1, 0, 0);
    glBegin(GL_TRIANGLES);
        glVertex2f(0, 0);
        glVertex2f(0, 0.5);
        glVertex2f(0.5, 0);
    glEnd();
}

void display(void)
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    //  Draw reference axes
    glColor3f(1, 1, 1);
    glBegin(GL_LINES);
        glVertex2f(-1, 0);  // start location
        glVertex2f(1, 0);   // end location
        glVertex2f(0, -1);  // start location
        glVertex2f(0, 1);   // end location
    glEnd();

    glPushMatrix();
        glTranslatef(t, 0, 0);  // the order of these transformations may affect the result
        glScalef(s, s, 0);      // because of the non-commutative features
        glRotatef(r, 0, 0, 1);  // here the rotation is applied first
        DrawTriangles();
    glPopMatrix();

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(width, height);
    glutCreateWindow("Transformations");

    glutDisplayFunc(display);
    glutIdleFunc(when_in_mainloop);
    glutKeyboardFunc(keyboard_input);

    glutMainLoop();
}
```

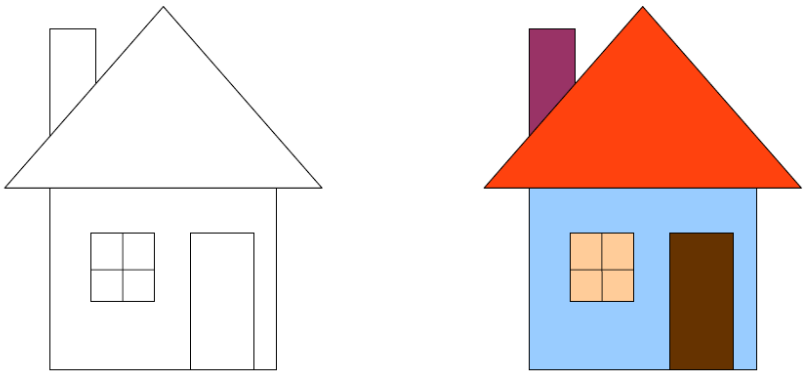
2) **Task 2**: Try to modify some of the parameters to see the effect.

3) **Task 3**: You should keep in mind how the functions work and can be called. You may now want to revisit the tasks and functions used.



4) Write a program that draws the house in two steps as show below.

Step 1: draw polygons
Step 2: fill in polygons with colour
Hints: mouse or keyboard interactions can be used to show the steps.



Part 2b - Geometric Transformations (Further Exercise)

1) Although matrix multiplication is associative (e.g., M3·M2·M1 = (M3·M2)·M1 = M3·(M2·M1)), transformation products are not always commutative (e.g., A·B ≠B·A).
Identify whether the following is true or false

A	B	A·B = B·A (True or False)
Translation	Translation	
Scaling	Scaling	
Rotation	Rotation	
Translation	Rotation	
Translation	Scaling	
Uniform scaling ($s_x = s_y$)	Translation	
Uniform scaling ($s_x = s_y$)	Rotation	

2) Show that the following sequences commute:

a) A rotation and a uniform scaling
b) Two rotations about the same axis
c) Two translations

3) Given the following transformation matrix, decide what transformation it is about

$$\begin{bmatrix} 1,0,0,0 \\ 0,\cos(t),\sin(t),0 \\ 0,-\sin(t),\cos(t),0 \\ 0,0,0,1 \end{bmatrix}$$

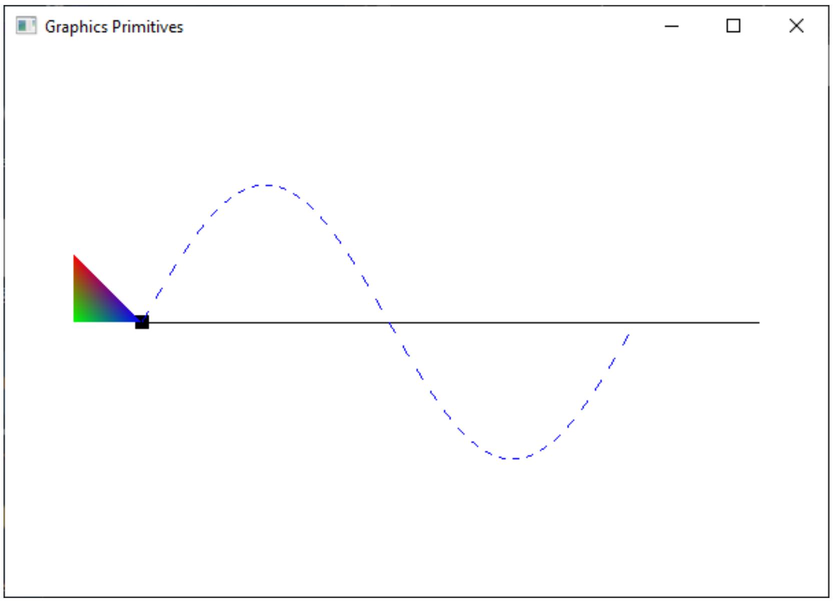
4) Applying standard scaling, your transformation results in changes in both the size and location of the object. How can you scale an object to its centre? For a rectangle with its bottom-left corner at (x1,y1) and top-right corner at (x2,y2), implement your idea in OpenGL.

5) The transformation that produces a mirror image is called reflection. How can you reflect an object (e.g., a triangle) about the x-axis, and the co-ordinate origin?

Part 3 – Additional Material

1) Sample program for Part I of Lab 03

```
// File ID: Lab03a.cpp
// Title:
// Author:

#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <Math.h>

void define_to_OpenGL();
///////////////////////////////////
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    // Task 2
    glutInitWindowSize(600, 400);
    glutInitWindowPosition(50, 50);

    glutCreateWindow("Graphics Primitives");
    glutDisplayFunc(define_to_OpenGL);
    glutMainLoop();
}
///////////////////////////////////
void define_to_OpenGL()
{
    glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    // The stuff to appear on screen goes here

    // Task 2
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100, 500, -200, 200);

    // Task 3
    glLineWidth(1.0);
    glColor3f(0, 0, 0);
    glBegin(GL_LINES);
    glVertex2f(0, 0); // start location
    glVertex2f(450, 0); // end location
    glEnd();

    // Task 4
    glPointSize(10);
    glColor3f(0, 0, 0);
    glBegin(GL_POINTS);
    glVertex2f(0, 0);
    glEnd();

    /* Task 5_1
    int i;
    float x,y;
    glColor3f(0.0,0.0,1.0);
    glPointSize(1);
    glBegin(GL_POINTS);
    for(i=0;i<361;i=i+5)
    {
        x = (float)i;
        y = 100.0 * sin(i*(6.284/360.0));
        glVertex2f(x,y);
    }
    glEnd(); */

    // Task 5_2
    int i;
    float x, y;
    glColor3f(0.0, 0.0, 1.0);
    glEnable(GL_LINE_STIPPLE);  // dashed line
    glLineStipple(2, 0x0F0F);   // 0X0F0F is a hexadecimal (hex) number, starting with 0X
    glBegin(GL_LINES);             // given pattern of 2 x 0000111100001111, draw 4x2 pixels,
    for (i = 0;i < 361;i = i + 5)  // skip 4*2 pixels, draw 4x2 pixels and skip 4x2 pixels,
    {
        x = (float)i;
        y = 100.0 * sin(i * (6.284 / 360.0));
        glVertex2f(x, y);
    }
    glEnd();
    glDisable(GL_LINE_STIPPLE);

    // Tasks 6, 7 and 8
    //glShadeModel(GL_SMOOTH);
    glBegin(GL_TRIANGLES);
    glColor3f(1, 0, 0);
    glVertex2f(-50, 50);
    glColor3f(0, 1, 0);
    glVertex2f(-50, 0);
    glColor3f(0, 0, 1);
    glVertex2f(0, 0);
    glEnd();
    glFlush();
}
//////////////////////////////////
```

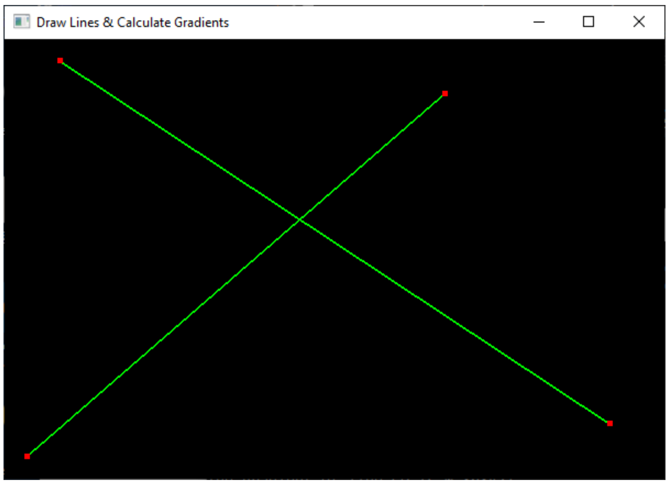


2) Sample program for drawing lines and calculating their gradients for Part 2.3 of Lab 02

Given line AB represented by points A(xA, yA) and B(xB,yB), and line CD represented by points C(xC, yC) and D(xD,yD), write a program that calls OpenGL to

a) draw AB and CD in the window you have created by altering the sample program in Part I, and using glBegin(GL_LINES)
b) calculate and output the lengths of the lines
c) calculate and output the gradients
d) check if the lines are perpendicular or parallel to each other
e) draw points A, B, C and D at the same time when the lines are drawn by calling glBegin(POINTS).

```
// File ID: Lab03b.cpp
// Title: Draw lines and calculate their gradients
// Author:

#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <Math.h>
#include <iostream>
#include "stdio.h"

using namespace std;

void define_to_OpenGL();
void init();

// Declare variables for storing coordinates of points A, B, C and D
float xa, ya, xb, yb, xc, yc, xd, yd;

///////////////////////////////////
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowSize(600, 400);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Draw Lines & Calculate Gradients");
    init();
    glutDisplayFunc(define_to_OpenGL);
    glutMainLoop();
}
///////////////////////////////////
void init() // Assign coordinate values of Points A, B, C and D
{
    cout << "To obtain a good display, \nthe values of X coordinates should be set between -100 and 500.
\nthe values of Y coordinates should be set between -200 and 200.\n";

    //Prompt to input and read in coordinate values
    cout << "\nPlease enter x and y coordinates of point A: ";
    cin >> xa >> ya;
    cout << "Please enter x and y coordinates of point B: ";
    cin >> xb >> yb;
    cout << "Please enter x and y coordinates of point C: ";
    cin >> xc >> yc;
    cout << "Please enter x and y coordinates of point D: ";
    cin >> xd >> yd;
}
///////////////////////////////////
void define_to_OpenGL()
{
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    // Set the projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Set orthographic projection parameters (left, right, bottpm, top)
    gluOrtho2D(-100, 500, -200, 200);

    // Calculate the length of the lines
    cout << "\nThe length of line AB is " << sqrtf(powf((xb - xa), 2.0) + powf((yb - ya), 2.0)) << ".\n";
    cout << "The length of line CD is " << sqrtf(powf((xd - xc), 2.0) + powf((yd - yc), 2.0)) << ".\n";

    // Calculate gradients of the lines
    cout << "\nThe gradient of line AB is " << (yb - ya) / (xb - xa) << ".";
    cout << "\nThe gradient of line CD is " << (yd - yc) / (xd - xc) << ".\n";

    // Check if the lines are perpendicular or parallel to each other
    if (((yb - ya) / (xb - xa)) * ((yd - yc) / (xd - xc)) == -1)
    {
        cout << "\nThe two lines are prependicular to each other.\n";
    }
    else if (((yb - ya) / (xb - xa)) == ((yd - yc) / (xd - xc)))
    {
        cout << "\nThe two lines are parallel to each other.\n";
    }
    else
    {
        cout << "\nThe two lines are neither perpendicular nor parallel to each other.\n";
    }

    glLineWidth(2.0);
    glColor3f(0, 1, 0);
    //Draw Line AB and CD
    glBegin(GL_LINES);
    glVertex2f(xa, ya);
    glVertex2f(xb, yb);

    glVertex2f(xc, yc);
    glVertex2f(xd, yd);
    glEnd();

    glPointSize(5);
    glColor3f(1, 0, 0);
    //Draw Points A, B, C and D
    glBegin(GL_POINTS);
    glVertex2f(xa, ya);
    glVertex2f(xb, yb);
    glVertex2f(xc, yc);
    glVertex2f(xd, yd);
    glEnd();

    glFlush();
}
```



Extra work: modify the code so that it can properly deal with lines that are aligned with the axes!