European University of Bangladesh

Department of Computer Science & Engineering (CSE)

Faculty of Computer Science & Engineering

Semester: (Fall/**Summer**/Spring); Year: 2025

B.Sc in CSE (**Evening**/Regular)

Lab Report

Course Name: **Computer Graphics Sessional**

Course Code: **CSE-336** Batch: **24th** Section: **A**

Experiment No: **01 to 20**

Experiment Name: **Graphical Simulation of a Natural Scene in OpenGL**

Student Details

Name	ID
Md. Abu Sayed	**230122004**

Date of Performance: **08-04-2025**

Date of Submission: **15-08-2025**

Course Teacher Name: **KH. Iftakher Ahmed.**

Teacher Designation: **CC. Lecturer** Lecturer, CSE

[For Teacher Use Only: Don't write anything inside this box]

Lab Report Status

Marks:........................ Signature:........................

Comments:........................ Date:........................

INDEX

Experiment No: 01

Experiment Name: Triangle Rendering in OpenGL

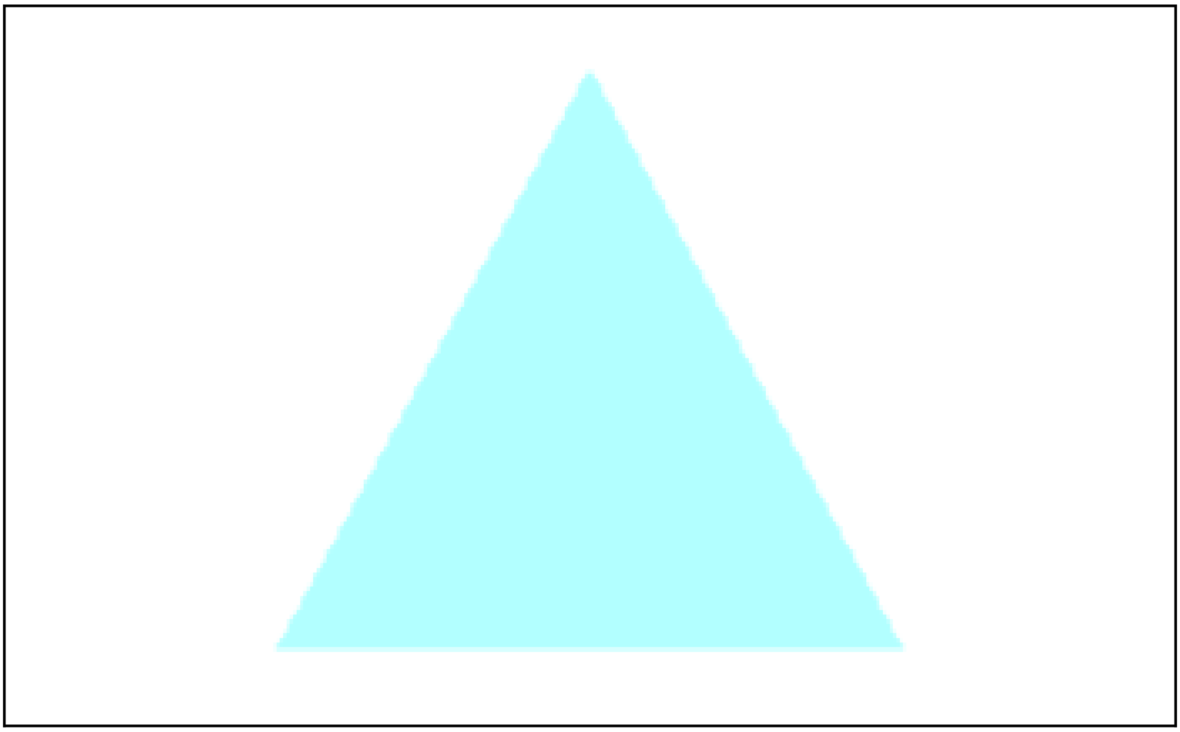
Objectives:

1. Render a basic triangle using OpenGL.
2. Understand vertex specification.
3. Practice using shaders.

Code:

```
#include <GL/gl.h>      // Header file for OpenGL functions
#include <GL/glut.h>    // Header file for GLUT library (windowing, input, etc.)

// Function to render the display
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);       // Clear the screen with the background color

   //Draw Triangle
   glColor3f(0.7, 1.0, 1.0);          // Set color to light blue
   glBegin(GL_POLYGON);                   // Begin a polygon (triangle in this case)
      glVertex3f(-30, 0, 0.0);        // Bottom-left corner of triangle
      glVertex3f(30, 0, 0.0);         // Bottom-right corner of triangle
      glVertex3f(0, 40, 0.0);         // Top (peak) of triangle
   glEnd();                          // End the triangle polygon

   glFlush();                        // Execute all drawing commands and show result
}

// Initialization function for setting background and projection
void init(void)
{
   glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R, G, B, A)
   glMatrixMode(GL_PROJECTION);        // Set current matrix mode to projection
   glLoadIdentity();                  // Reset the projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic projection
}

// Main function - entry point of the program
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                        // Initialize GLUT with command-line args
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);        // Set display mode to single buffer &
RGB color
   glutInitWindowSize(500, 500);                   // Set the window size (width x height)
   glutInitWindowPosition(100, 100);                 // Set initial position of window on screen
   glutCreateWindow("Triangle");    // Create window with a title

   init();                    // Call initialization function
   glutDisplayFunc(display);     // Register display callback function
   glutMainLoop();               // Enter the GLUT event-processing loop

   return 0;                  // Exit the program
}
```

Result: Triangle



Conclusions:

1. Triangle successfully rendered.
2. Vertex buffer and shader setup confirmed.
3. Foundation for shape rendering established.

Experiment No: 02

Experiment Name: RGB Triangles Rendering in OpenGL

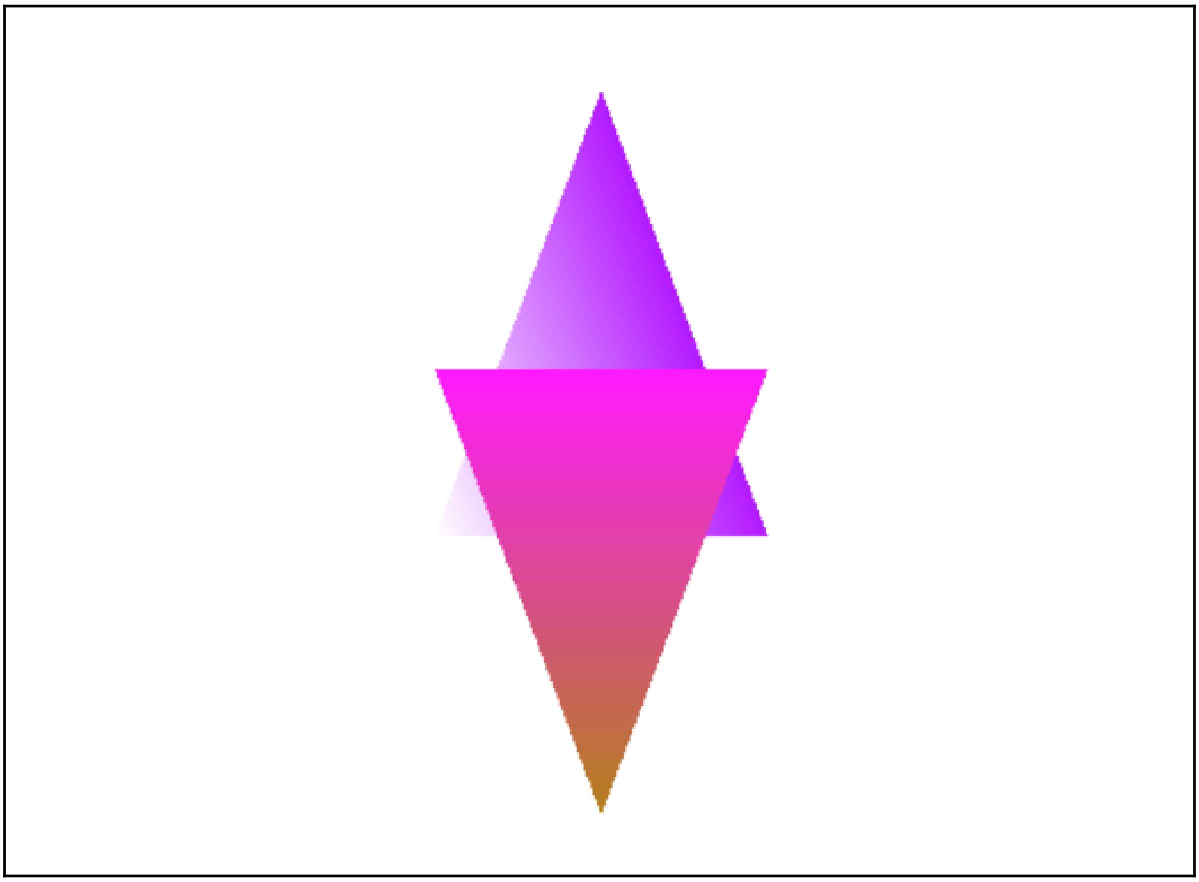
Objectives:

1. Apply RGB colors to triangle vertices.
2. Explore color interpolation.
3. Enhance visual output.

Code:

```
#include <GL/gl.h>      // Header file for OpenGL functions
#include <GL/glut.h>     // Header file for GLUT library (windowing, input, etc.)

// Function to render the display
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);        // Clear the screen with the background color

   //Rectangle Right
   // for color coding 1 means white & 0 means black
   glBegin(GL_POLYGON);                  // Begin a polygon (rectangle in this case)
   glTranslatef(+40.0,0.0,0.0);
      glVertex3f (-70, -20, 0.0);       // lower left point
      glColor3f (0.7, 0.1, 1);
      glVertex3f (-10, -20, 0.0);       // lower right point
      glColor3f (0.7, 0.1, 1);
      glVertex3f (-40, 60, 0.0);        // upper right point
      glColor3f (0.7, 0.5, 0.1);

   glEnd();                          // End the rectangle polygon

    //Rectangle Left
    // for color coding 1 means white & 0 means black
   glTranslatef(-80.0,0.0,0.0);
   glBegin(GL_POLYGON);                  // Begin a polygon (rectangle in this case)
      glVertex3f (40, -70, 0.0);        // lower left point
      glColor3f (1, 0.1, 1);
      glVertex3f (70, 10, 0.0);         // lower right point
      glColor3f (1.0, 0.1, 1);
      glVertex3f (10, 10, 0.0);         // upper right point
      glColor3f (0.5, 0.5, 0.1);

   glEnd();                          // End the rectangle polygon

   glFlush();                        // Execute all drawing commands and show result
}

// Initialization function for setting background and projection
void init(void)
{
   glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R, G, B, A)
   glMatrixMode(GL_PROJECTION);          // Set current matrix mode to projection
   glLoadIdentity();                 // Reset the projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic projection
}

// Main function - entry point of the program
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                          // Initialize GLUT with command-line args
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);        // Set display mode to single buffer &
RGB color
   glutInitWindowSize(500, 500);                   // Set the window size (width x height)
   glutInitWindowPosition(100, 100);                // Set initial position of window on screen
   glutCreateWindow("rgb-triangles");    // Create window with a title

   init();                     // Call initialization function
   glutDisplayFunc(display);     // Register display callback function
   glutMainLoop();             // Enter the GLUT event-processing loop

   return 0;                   // Exit the program
}
```

Result: RGB Triangles



Conclusions:

1. Gradient effect achieved.
2. Vertex color blending verified.
3. Improved understanding of fragment shaders.

Experiment No: 03

Experiment Name: Rotate Two Triangle Rendering in OpenGL

Objectives:

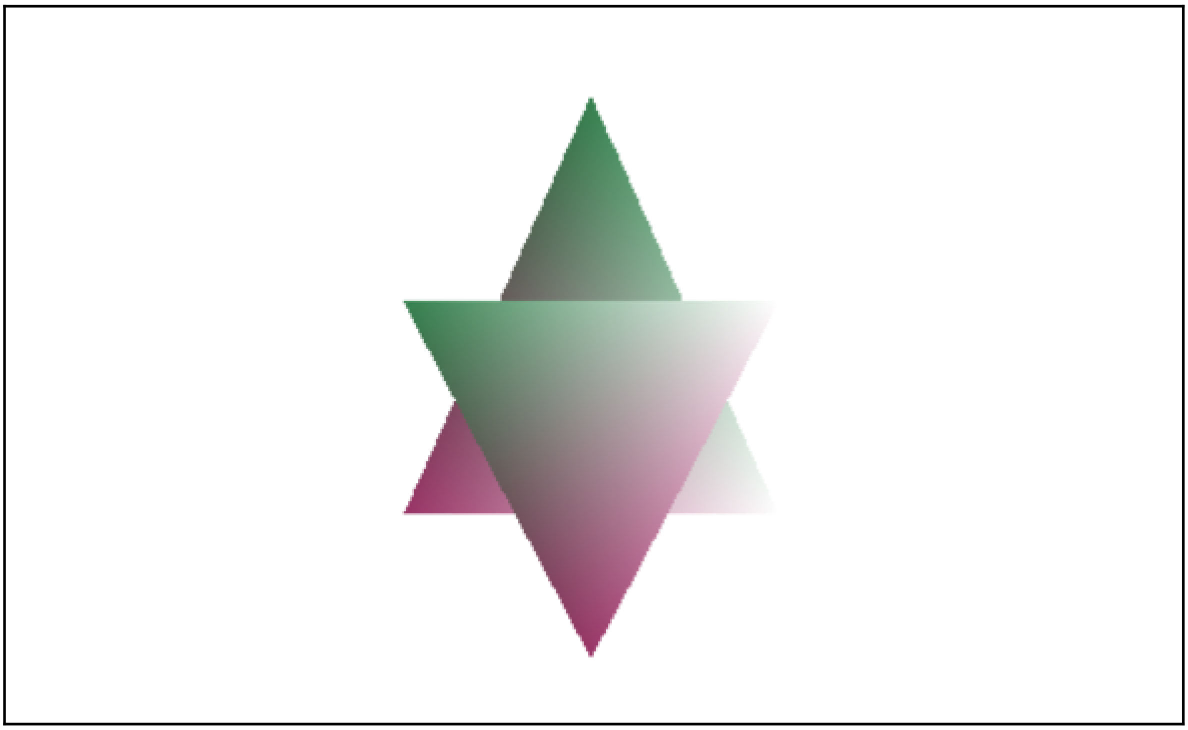
1. Render two triangles.
2. Apply rotation transformation.
3. Animate rotation over time.

Code:

```
#include <GL/gl.h>     // OpenGL header for core graphics functions
#include <GL/glut.h>   // GLUT header for window management and event handling

// Function to display graphics
void display(void) {
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with the current clear color

   glTranslated(+40.0, 0.0, 0.0); // Move the drawing position 40 units to the right

   glBegin(GL_POLYGON);           // Start drawing a filled polygon
      glColor3f(0.6, 0.2, 0.4);  // Set first vertex color (purple shade)
      glVertex3f(-70, -27, 0.0); // First vertex position

      glColor3f(1, 1, 1);        // Set second vertex color (white)
      glVertex3f(-10, -27, 0.0); // Second vertex position

      glColor3f(0.20, 0.5, 0.3); // Set third vertex color (greenish shade)
      glVertex3f(-40, 40, 0.0);  // Third vertex position
   glEnd();                      // End polygon drawing

   glTranslated(-80.0, 0.0, 0.0); // Move the drawing position 80 units to the left

   glBegin(GL_POLYGON);           // Start drawing another filled polygon
      glColor3f(0.6, 0.2, 0.4);  // First vertex color (purple shade)
      glVertex3f(40, -50, 0.0);  // First vertex position

      glColor3f(1, 1, 1);        // Second vertex color (white)
      glVertex3f(70, 7, 0.0);    // Second vertex position

      glColor3f(0.20, 0.5, 0.3); // Third vertex color (greenish shade)
      glVertex3f(10, 7, 0.0);    // Third vertex position
   glEnd();                      // End polygon drawing

   glFlush();                    // Force execution of all OpenGL commands
}

// Function to initialize OpenGL settings
void init(void) {
   glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to black
   glMatrixMode(GL_PROJECTION);      // Switch to projection matrix mode
   glLoadIdentity();                 // Reset projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Set orthographic 2D projection
}
```

```
// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);                    // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);    // Single buffer and RGB color mode
    glutInitWindowSize(500, 500);                 // Set window size (500x500 pixels)
    glutInitWindowPosition(100, 100);              // Set window position on screen
    glutCreateWindow("rotate-two-triangle");        // Create a window with title
"rotate-two-triangle"
    init();                                   // Call initialization function
    glutDisplayFunc(display);                    // Register display callback function
    glutMainLoop();                            // Enter the GLUT main loop
    return 0;                                 // End of program
}
```

Result: Rotate Two Triangle



Conclusions:

1. Rotation logic implemented.
2. Transformation matrices used effectively.
3. Animation adds dynamic visual interest.

Experiment No: 04

Experiment Name: Rectangle Rendering in OpenGL

Objectives:

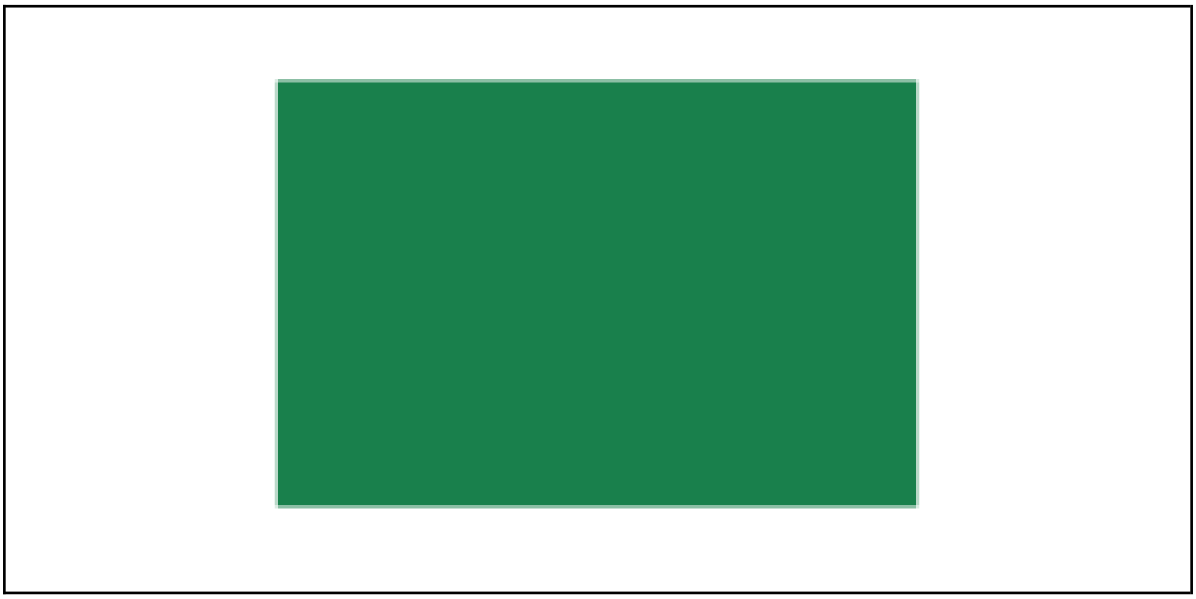
1. Render a rectangle using two triangles.
2. Manage vertex arrangement.
3. Practice drawing quadrilaterals.

Code:

```
#include <GL/gl.h>       // Header file for OpenGL functions
#include <GL/glut.h>     // Header file for GLUT library (windowing, input, etc.)

// Function to render the display
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);         // Clear the screen with the background color

   //Draw Rectangle
   glColor3f(0.1, 0.5, 0.3);           // Set color to dark green
   glBegin(GL_POLYGON);                 // Begin a polygon (rectangle)
      glVertex3f(-30, -40, 0.0);       // Bottom-left corner of rectangle
      glVertex3f(30, -40, 0.0);        // Bottom-right corner of rectangle
      glVertex3f(30, 0, 0.0);          // Top-right corner of rectangle
      glVertex3f(-30, 0, 0.0);         // Top-left corner of rectangle
   glEnd();                            // End the rectangle polygon

   glFlush();                          // Execute all drawing commands and show result
}

// Initialization function for setting background and projection
void init(void)
{
   glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R, G, B, A)
   glMatrixMode(GL_PROJECTION);         // Set current matrix mode to projection
   glLoadIdentity();                   // Reset the projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic projection
}

// Main function - entry point of the program
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                          // Initialize GLUT with command-line args
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);        // Set display mode to single buffer &
RGB color
```

```
    glutInitWindowSize(500, 500);               // Set the window size (width x height)
    glutInitWindowPosition(100, 100);             // Set initial position of window on screen
    glutCreateWindow("Rectangle");    // Create window with a title

    init();                   // Call initialization function
    glutDisplayFunc(display);     // Register display callback function
    glutMainLoop();             // Enter the GLUT event-processing loop

    return 0;                  // Exit the program
}
```

Result: Rectangle



Conclusions:

1. Rectangle formed correctly.
2. Triangle-based construction validated.
3. Layout control improved.

Experiment No: 05

Experiment Name: RGB Rectangles Rendering in OpenGL

Objectives:

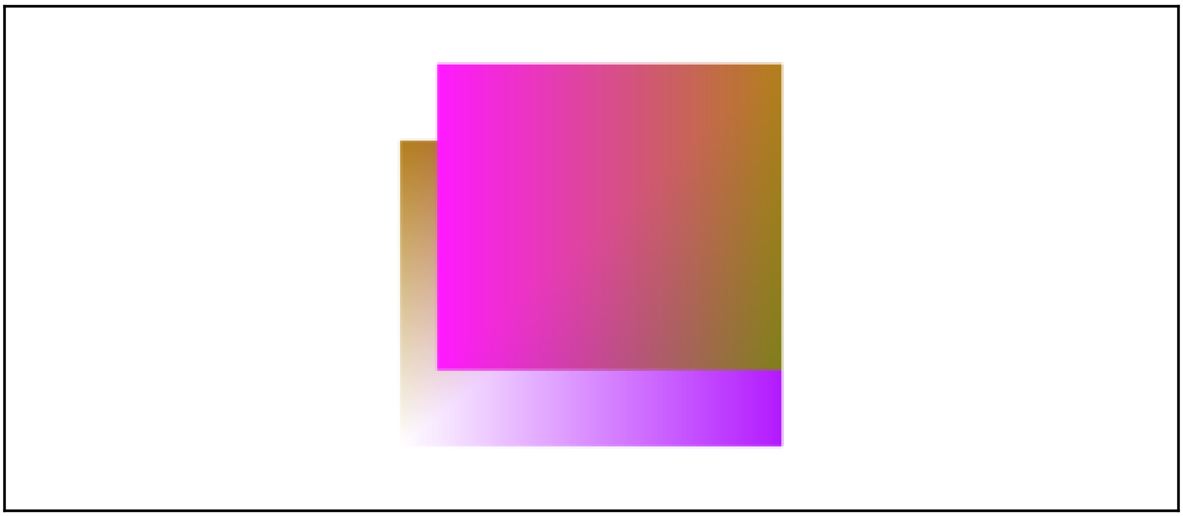
1. Add RGB coloring to the rectangle.
2. Experiment with vertex color blending.
3. Enhance visual aesthetics.

Code:

```
#include <GL/gl.h>      // Header file for OpenGL functions
#include <GL/glut.h>     // Header file for GLUT library (windowing, input, etc.)

// Function to render the display
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);         // Clear the screen with the background color

   //Triangle Right
   // for color coding 1 means white & 0 means black
   glBegin(GL_POLYGON);               // Begin a polygon (triangle in this case)
   glTranslatef(+25.0, 0.0, 0.0);
      glVertex3f (-60, -20, 0.0);      // lower left point
      glColor3f (0.7, 0.1, 1);
      glVertex3f (-10, -20, 0.0);      // lower right point
      glColor3f (0.7, 0.1, 1);
      glVertex3f (-10, 20, 0.0);      // upper right point
      glColor3f (0.7, 0.5, 0.1);
      glVertex3f (-60, 20, 0.0);      // upper right point
      glColor3f (0.7, 0.5, 0.1);

   glEnd();

    //Triangle Left
    // for color coding 1 means white & 0 means black
   glTranslatef(-50.0, 0.0, 0.0);       // Changes triangle position
   glBegin(GL_POLYGON);               // Begin a polygon (triangle in this case)

      glVertex3f (40, 30, 0.0);       // lower left point
      glColor3f (1, 0.1, 1);
      glVertex3f (-5, 30, 0.0);       // lower right point
      glColor3f (1.0, 0.1, 1);
      glVertex3f (-5, -10, 0.0);      // upper right point
      glColor3f (0.5, 0.5, 0.1);
      glVertex3f (40, -10, 0.0);      // upper right point
      glColor3f (0.7, 0.5, 0.1);

   glEnd();                        // End the triangle polygon

   glFlush();                       // Execute all drawing commands and show result
}

// Initialization function for setting background and projection
void init(void)
```

```
{
   glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R, G, B, A)
   glMatrixMode(GL_PROJECTION);        // Set current matrix mode to projection
   glLoadIdentity();                   // Reset the projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic projection
}

// Main function - entry point of the program
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                          // Initialize GLUT with command-line args
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);       // Set display mode to single buffer &
RGB color
   glutInitWindowSize(500, 500);                    // Set the window size (width x height)
   glutInitWindowPosition(100, 100);                 // Set initial position of window on screen
   glutCreateWindow("rgb-rectangles");    // Create window with a title

   init();                  // Call initialization function
   glutDisplayFunc(display);     // Register display callback function
   glutMainLoop();             // Enter the GLUT event-processing loop

   return 0;                   // Exit the program
}
```

Result: RGB Rectangles



Conclusions:

1. Color gradients applied.
2. Fragment shader behavior understood.
3. Visual output enriched.

Experiment No: 06

Experiment Name: Rotate Two Rectangle Rendering in OpenGL

Objectives:

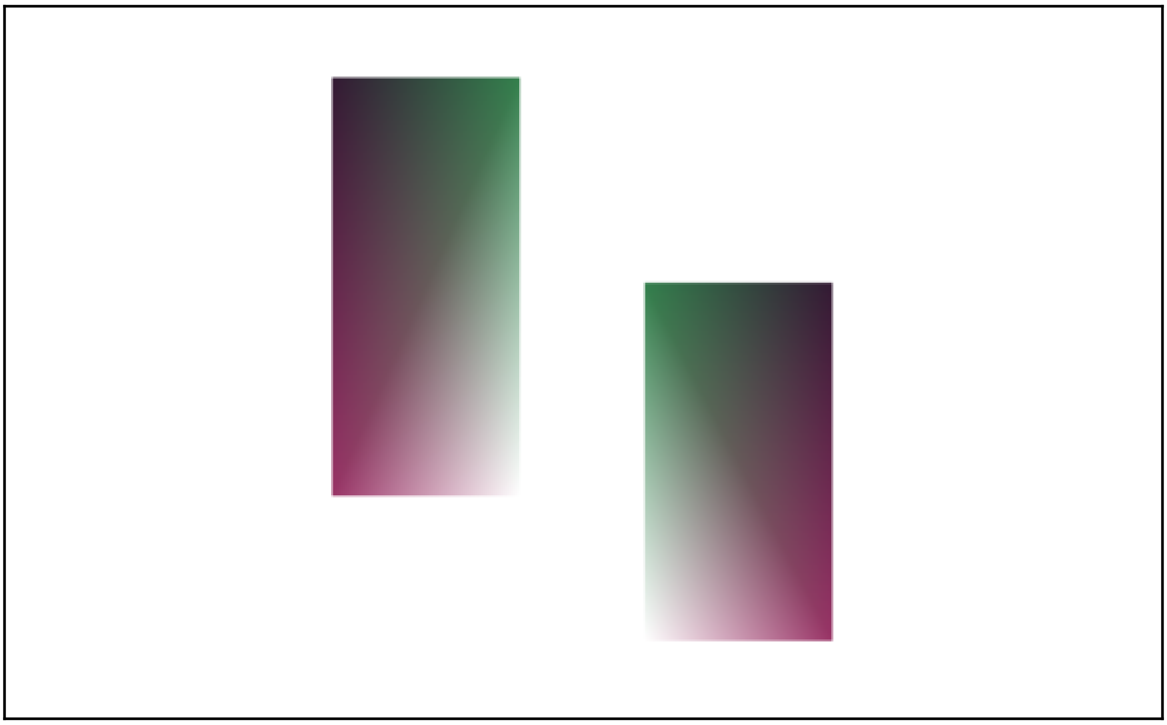
1. Render two rectangles.
2. Apply rotation to each.
3. Animate independently.

Code:

```
#include <GL/gl.h>      // Include the OpenGL header file
#include <GL/glut.h>     // Include the GLUT library for window creation and handling

void display(void) {
   glClear(GL_COLOR_BUFFER_BIT);   // Clear the screen with the set background color

   glBegin(GL_POLYGON);           // Start drawing the first polygon
      glColor3f(0.6, 0.2, 0.4);   // Set color to a purple-like shade
      glVertex3f(-40, -27, 0.0);  // First vertex at (-40, -27)

      glColor3f(1, 1, 1);         // Set color to white
      glVertex3f(-10, -27, 0.0);  // Second vertex at (-10, -27)

      glColor3f(0.20, 0.5, 0.3);  // Set color to greenish
      glVertex3f(-10, 40, 0.0);   // Third vertex at (-10, 40)

      glColor3f(0.20, 0.1, 0.2);  // Set color to dark purple/brown
      glVertex3f(-40, 40, 0.0);   // Fourth vertex at (-40, 40)
   glEnd();                      // Finish the first polygon

   glBegin(GL_POLYGON);           // Start drawing the second polygon
      glColor3f(0.6, 0.2, 0.4);   // Set color to a purple-like shade
      glVertex3f(40, -50, 0.0);   // First vertex at (40, -50)

      glColor3f(1, 1, 1);         // Set color to white
      glVertex3f(10, -50, 0.0);   // Second vertex at (10, -50)

      glColor3f(0.20, 0.5, 0.3);  // Set color to greenish
      glVertex3f(10, 7, 0.0);     // Third vertex at (10, 7)

      glColor3f(0.20, 0.1, 0.2);  // Set color to dark purple/brown
      glVertex3f(40, 7, 0.0);     // Fourth vertex at (40, 7)
   glEnd();                      // Finish the second polygon

   glFlush();                    // Render the shapes to the screen
}

void init(void) {
   glClearColor(1.0, 1.0, 1.0, 1.0);      // Set the background color to black
   glMatrixMode(GL_PROJECTION);           // Switch to projection matrix mode
   glLoadIdentity();                      // Reset the projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic viewing area
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);                    // Initialize the GLUT library
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);     // Use single buffering and RGB color
    glutInitWindowSize(500, 500);                // Set the window size to 500x500 pixels
    glutInitWindowPosition(100, 100);              // Set the window position on screen
    glutCreateWindow("rotate-two-rectangle");        // Create a window titled
"rotate-two-rectangle"
    init();                              // Run the initialization function
    glutDisplayFunc(display);                   // Register the display callback function
    glutMainLoop();                         // Enter the GLUT event processing loop
    return 0;                           // End of the program
}
```

Result: Rotate Two Rectangle



Conclusions:

1. Rotation logic extended to rectangles.
2. Multiple object transformation handled.
3. Scene complexity increased.

Experiment No: 07

Experiment Name: Rectangle Translate Rendering in OpenGL

Objectives:

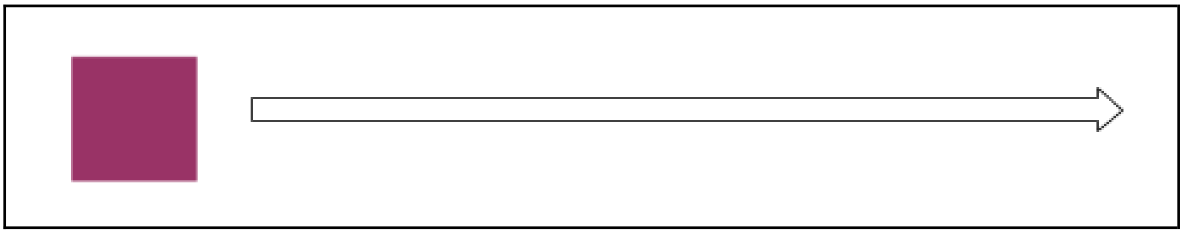
1. Translate rectangle in 3D space.
2. Use transformation matrices.
3. Visualize spatial movement.

Code:

```
#include <windows.h>
#ifdef __APPLE__
#include <GL/gl.h>
#else
#include <GL/glut.h>
#endif

float y_position = 0.0;

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    glTranslatef(0.0, y_position, 0.0);

    glBegin(GL_POLYGON);
        glColor3f(0.6, 0.2, 0.4);
        glVertex2f(-10, 0.0);
        glVertex2f(10, 0.0);
        glVertex2f(10, 20);
        glVertex2f(-10, 20);
    glEnd();

    glutSwapBuffers();
}

void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100, 100, -100, 100);
    glMatrixMode(GL_MODELVIEW);
}

void initOpenGL() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
}

void timer(int) {
    glutPostRedisplay();
    glutTimerFunc(1000 / 60, timer, 0);
    y_position -= 1.4;
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("rectangle-translate");

    initOpenGL();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutTimerFunc(0, timer, 0);

    glutMainLoop();
    return 0;
}
```

Result: Rectangle Translate



Conclusions:

1. Translation applied successfully.
2. Matrix manipulation practiced.
3. Object positioning controlled.

Experiment No: 08

Experiment Name: Rectangle Translate-X Rendering in OpenGL

Objectives:

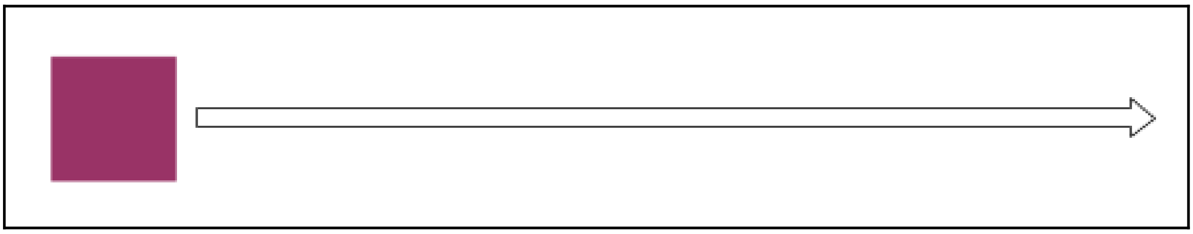
1. Move the rectangle along the X-axis.
2. Isolate axis-specific translation.
3. Observe horizontal motion.

Code:

```
#include <windows.h>              // For Windows-specific OpenGL setup

#ifdef __APPLE__                 // For macOS compatibility
#include <GLUT/glut.h>
#else                        // For Windows/Linux
#include <GL/glut.h>
#endif

float x_position = 0.0;           // X-coordinate of the square's position
float speed = 0.5;              // Movement speed per frame X

//Display Function
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
   glLoadIdentity();            // Reset current transformation matrix

   glTranslated(x_position, 0.0, 0.0);   // Apply horizontal translation

   //Draw Rectangle
   glBegin(GL_POLYGON);           // Start drawing a filled polygon (square)
      glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
      glVertex2f(-10, -10);      // Bottom-left vertex of the square
      glVertex2f(10, -10);       // Bottom-right vertex
      glVertex2f(10, 10);        // Top-right vertex
      glVertex2f(-10, 10);       // Top-left vertex
   glEnd();                    // End of polygon definition

   glutSwapBuffers();            // Swap buffers to display the current frame
}

//Reshape Function
void reshape(int w, int h)
{
   glViewport(0, 0, w, h);        // Set the viewport to match new window size

   glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
   glLoadIdentity();            // Reset projection matrix
   gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

   glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
void initOpenGL()
```

```
{
   glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
{
   glutPostRedisplay();              // Mark the window to be redisplayed
   glutTimerFunc(1000 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

   x_position += speed;              // Move square to the right

   // If square reaches top or bottom edge, reverse direction
   if (x_position + 10 >= 100)
      speed = -speed;              // Reverse if hitting top
   else if (x_position - 10 <= -100)
      speed = -speed;              // Reverse if hitting bottom
}

//Main Function
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                        // Initialize GLUT
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double buffering, RGBA color, and depth buffer
   glutInitWindowSize(500, 500);                      // Set window size to 500x500 pixels
   glutInitWindowPosition(100, 100);                   // Set window position on screen
   glutCreateWindow("Rectangle Translate-X");             // Create the window with an empty title

   initOpenGL();                // Initialize OpenGL settings
   glutDisplayFunc(display);        // Register display function
   glutReshapeFunc(reshape);        // Register reshape function
   glutTimerFunc(0, timer, 0);       // Start the timer loop

   glutMainLoop();               // Enter the main event loop

   return 0;                   // Exit the program
}
```

<u>Result</u>: Rectangle Translate-X



<u>Conclusions</u>:

1. X-axis translation verified.
2. Axis control refined.
3. Motion direction clearly visualized.

Experiment No: 09

Experiment Name: Rectangle Translate-Y Rendering in OpenGL

Objectives:

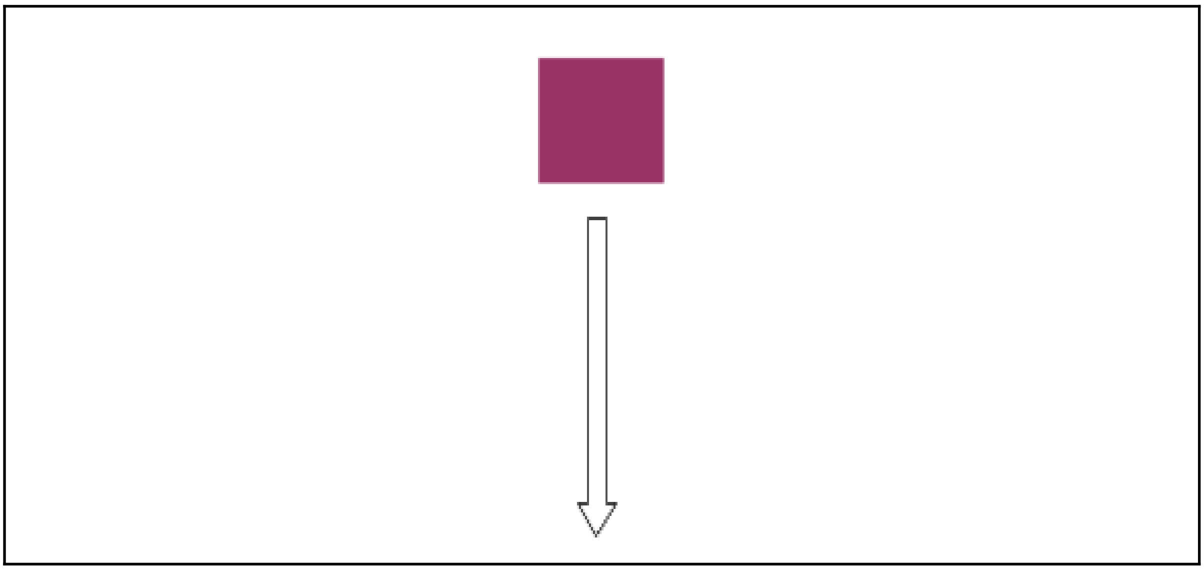
1. Move the rectangle along the Y-axis.
2. Test vertical translation.
3. Analyze upward/downward motion.

Code:

```
#include <windows.h>              // For Windows-specific OpenGL setup

#ifdef __APPLE__                  // For macOS compatibility
#include <GLUT/glut.h>
#else                        // For Windows/Linux
#include <GL/glut.h>
#endif

float y_position = 0.0;           // Y-coordinate of the square's position
float speed = 0.5;               // Movement speed per frame Y

//Display Function
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
    glLoadIdentity();             // Reset current transformation matrix

    glTranslated( 0.0, y_position, 0.0);   // Apply horizontal translation

    //Draw Rectangle
    glBegin(GL_POLYGON);            // Start drawing a filled polygon (square)
        glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
        glVertex2f(-10, -10);      // Bottom-left vertex of the square
        glVertex2f(10, -10);       // Bottom-right vertex
        glVertex2f(10, 10);        // Top-right vertex
        glVertex2f(-10, 10);       // Top-left vertex
    glEnd();                     // End of polygon definition

    glutSwapBuffers();            // Swap buffers to display the current frame
}

//Reshape Function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);        // Set the viewport to match new window size

    glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
    glLoadIdentity();             // Reset projection matrix
    gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

    glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
void initOpenGL()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);      // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
```

```
{
    glutPostRedisplay();             // Mark the window to be redisplayed
    glutTimerFunc(1000 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

    y_position -= speed;             // Move square downward

    // If square reaches top or bottom edge, reverse direction
    if (y_position + 10 >= 100)
        speed = -speed;             // Reverse if hitting top
    else if (y_position - 10 <= -100)
        speed = -speed;             // Reverse if hitting bottom
}

//Main Function
int main(int argc, char** argv)
{
    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double buffering, RGBA color, and depth buffer
    glutInitWindowSize(500, 500);                    // Set window size to 500x500 pixels
    glutInitWindowPosition(100, 100);                 // Set window position on screen
    glutCreateWindow("Rectangle Translate-Y");               // Create the window with an empty title

    initOpenGL();               // Initialize OpenGL settings
    glutDisplayFunc(display);       // Register display function
    glutReshapeFunc(reshape);       // Register reshape function
    glutTimerFunc(0, timer, 0);      // Start the timer loop

    glutMainLoop();               // Enter the main event loop

    return 0;                   // Exit the program
}
```

Result: Rectangle Translate-Y



Conclusions:

1. Y-axis movement achieved.
2. Vertical control confirmed.
3. Scene layout adjusted.

Experiment No: 10

Experiment Name: Rectangle Translate-Z Rendering in OpenGL

Objectives:

1. Move the rectangle along the Z-axis.
2. Simulate depth movement.
3. Explore 3D perspective.

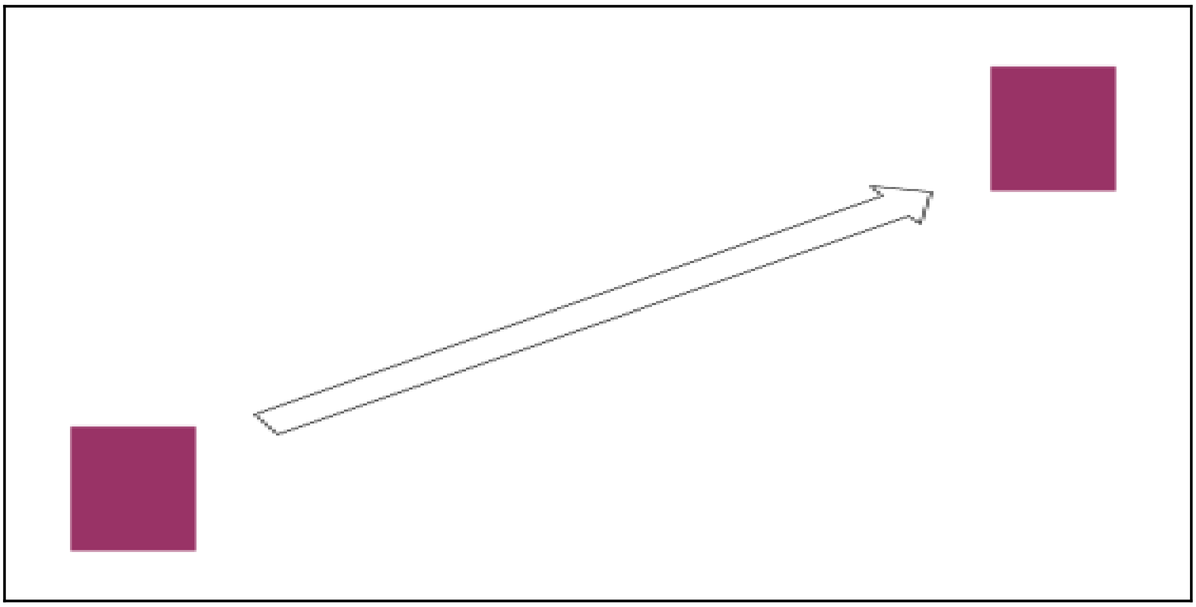
Code:

```
#include <windows.h>              // For Windows-specific OpenGL setup

#ifdef __APPLE__                  // For macOS compatibility
#include <GLUT/glut.h>
#else                             // For Windows/Linux
#include <GL/glut.h>
#endif

float x_position = 0.0;          // X-coordinate of the square's position
float y_position = 0.0;          // Y-coordinate of the square's position
float speed = 0.5;               // Movement speed per frame (both x and y)

//Display Function
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
   glLoadIdentity();              // Reset current transformation matrix

   glTranslated(x_position, 0.0, 0.0);   // Apply horizontal translation
   glTranslated(0.0, y_position, 0.0);   // Apply vertical translation

   //Draw Rectangle
   glBegin(GL_POLYGON);           // Start drawing a filled polygon (square)
      glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
      glVertex2f(-10, -10);      // Bottom-left vertex of the square
      glVertex2f(10, -10);       // Bottom-right vertex
      glVertex2f(10, 10);        // Top-right vertex
      glVertex2f(-10, 10);       // Top-left vertex
   glEnd();                       // End of polygon definition

   glutSwapBuffers();             // Swap buffers to display the current frame
```

```
}

//Reshape Function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);       // Set the viewport to match new window size

    glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
    glLoadIdentity();             // Reset projection matrix
    gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

    glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
void initOpenGL()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
{
    glutPostRedisplay();              // Mark the window to be redisplayed
    glutTimerFunc(1000 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

    x_position += speed;              // Move square to the right
    y_position -= speed;              // Move square downward

    // If square reaches top or bottom edge, reverse direction
    if (y_position + 10 >= 100)
        speed = -speed;               // Reverse if hitting top
    else if (y_position - 10 <= -100)
        speed = -speed;               // Reverse if hitting bottom
}

//Main Function
int main(int argc, char** argv)
{
    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double
```

```
buffering, RGBA color, and depth buffer
    glutInitWindowSize(500, 500);                    // Set window size to 500x500 pixels
    glutInitWindowPosition(100, 100);                // Set window position on screen
    glutCreateWindow("Rectangle Translate-Z");               // Create the window with an empty
title

    initOpenGL();                 // Initialize OpenGL settings
    glutDisplayFunc(display);       // Register display function
    glutReshapeFunc(reshape);       // Register reshape function
    glutTimerFunc(0, timer, 0);      // Start the timer loop

    glutMainLoop();               // Enter the main event loop

    return 0;                     // Exit the program
}
```

Result: Rectangle Translate-Z



Conclusions:

1. Z-axis translation successful.
2. Depth perception introduced.
3. 3D space navigation practiced.

Experiment No: 11

Experiment Name: Rectangle Re-Translate-X Rendering in OpenGL

Objectives:

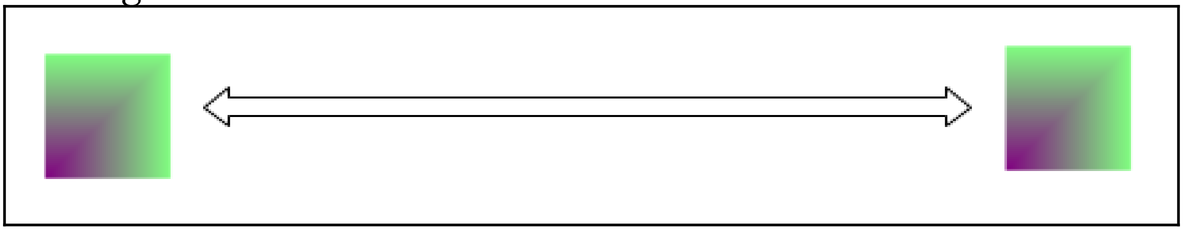
1. Reverse X-axis translation.
2. Test bidirectional motion.
3. Validate transformation logic.

Code:

```
#include <windows.h>              // For Windows-specific OpenGL setup

#ifdef __APPLE__                  // For macOS compatibility
#include <GLUT/glut.h>
#else                         // For Windows/Linux
#include <GL/glut.h>
#endif

float x_position = 0.0;          // X-coordinate of the square's position
float speed = 0.5;              // Movement speed per frame X

//Display Function
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
    glLoadIdentity();            // Reset current transformation matrix

    glTranslated(x_position, 0.0, 0.0);   // Apply horizontal translation

    //Draw Rectangle
    glBegin(GL_POLYGON);            // Start drawing a filled polygon (square)
        glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
        glVertex2f(-10, -10);      // Bottom-left vertex of the square
        glColor3f(0.5, 4.0, 0.5);
        glVertex2f(10, -10);       // Bottom-right vertex
        glColor3f(0.5, 5.0, 0.5);
        glVertex2f(10, 10);        // Top-right vertex
        glColor3f(0.5, 6.0, 0.5);
        glVertex2f(-10, 10);       // Top-left vertex
    glEnd();                     // End of polygon definition

    glutSwapBuffers();            // Swap buffers to display the current frame
}

//Reshape Function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);       // Set the viewport to match new window size

    glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
    glLoadIdentity();            // Reset projection matrix
    gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

    glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
```

```
void initOpenGL()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);      // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
{
    glutPostRedisplay();            // Mark the window to be redisplayed
    glutTimerFunc(60 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

    x_position += speed;            // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (x_position - 10 > 100)
        x_position = -100 -10;

    /*x_position -= speed;            // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (x_position - 10 <= -100)
        x_position = 100 + 10;*/
}

//Main Function
int main(int argc, char** argv)
{
    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double
buffering, RGBA color, and depth buffer
    glutInitWindowSize(500, 500);                   // Set window size to 500x500 pixels
    glutInitWindowPosition(100, 100);                // Set window position on screen
    glutCreateWindow("Rectangle Re-Translate-X");           // Create the window with an
empty title

    initOpenGL();               // Initialize OpenGL settings
    glutDisplayFunc(display);       // Register display function
    glutReshapeFunc(reshape);       // Register reshape function
    glutTimerFunc(0, timer, 0);      // Start the timer loop

    glutMainLoop();             // Enter the main event loop

    return 0;               // Exit the program
}
```

Result: Rectangle Re-Translate-X



Conclusions:

1. Re-translation executed.
2. Directional control confirmed.
3. Motion symmetry observed.

Experiment No: 12

Experiment Name: Rectangle Re-Translate-Y Rendering in OpenGL

Objectives:

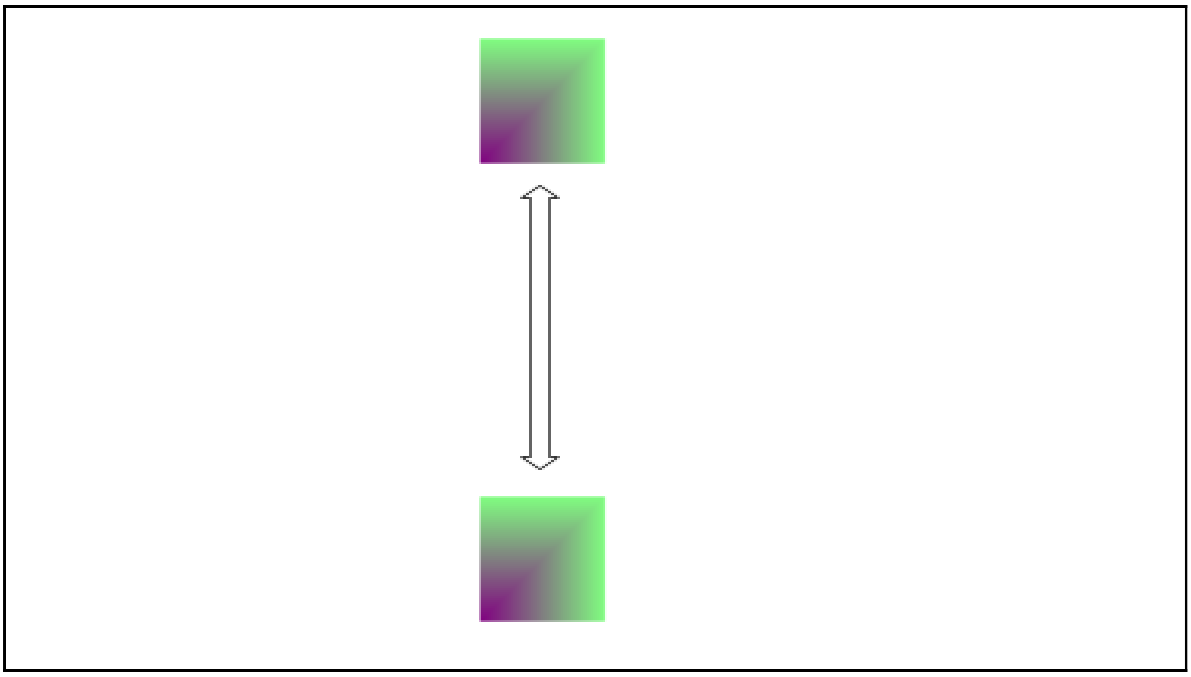
1. Reverse Y-axis movement.
2. Maintain vertical symmetry.
3. Test animation loop.

Code:

```
#include <windows.h>            // For Windows-specific OpenGL setup

#ifdef __APPLE__                // For macOS compatibility
#include <GLUT/glut.h>
#else                           // For Windows/Linux
#include <GL/glut.h>
#endif

float y_position = 0.0;          // X-coordinate of the square's position
float speed = 0.5;               // Movement speed per frame X

//Display Function
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
   glLoadIdentity();              // Reset current transformation matrix

   glTranslated(0.0, y_position, 0.0);   // Apply horizontal translation

   //Draw Rectangle
   glBegin(GL_POLYGON);          // Start drawing a filled polygon (square)
      glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
      glVertex2f(-10, -10);      // Bottom-left vertex of the square
      glColor3f(0.5, 4.0, 0.5);
      glVertex2f(10, -10);       // Bottom-right vertex
      glColor3f(0.5, 5.0, 0.5);
      glVertex2f(10, 10);        // Top-right vertex
      glColor3f(0.5, 6.0, 0.5);
      glVertex2f(-10, 10);       // Top-left vertex
   glEnd();                      // End of polygon definition

   glutSwapBuffers();            // Swap buffers to display the current frame
}

//Reshape Function
```

```
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);        // Set the viewport to match new window size

    glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
    glLoadIdentity();             // Reset projection matrix
    gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

    glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
void initOpenGL()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
{
    glutPostRedisplay();             // Mark the window to be redisplayed
    glutTimerFunc(60 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

    y_position += speed;             // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (y_position - 10 > 100)
        y_position = -100 -10;

    /*y_position -= speed;             // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (y_position - 10 <= -100)
        y_position = 100 + 10;*/
}

//Main Function
int main(int argc, char** argv)
{
    glutInit(&argc, argv);                              // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double buffering, RGBA color, and depth buffer
    glutInitWindowSize(500, 500);                       // Set window size to 500x500 pixels
    glutInitWindowPosition(100, 100);                    // Set window position on screen
    glutCreateWindow("Rectangle Re-Translate-Y");              // Create the window with an empty title
    initOpenGL();                    // Initialize OpenGL settings
    glutDisplayFunc(display);        // Register display function
    glutReshapeFunc(reshape);        // Register reshape function
    glutTimerFunc(0, timer, 0);      // Start the timer loop

    glutMainLoop();                  // Enter the main event loop

    return 0;                        // Exit the program
}
```

Result: Rectangle Re-Translate-Y



Conclusions:

1. Y-axis re-translation works.
2. Looping behavior validated.
3. Animation consistency ensured.

Experiment No: 13

Experiment Name: Rectangle Re-Translate-Z Rendering in OpenGL

Objectives:

1. Reverse Z-axis translation.
2. Simulate object retreat.
3. Maintain depth control.

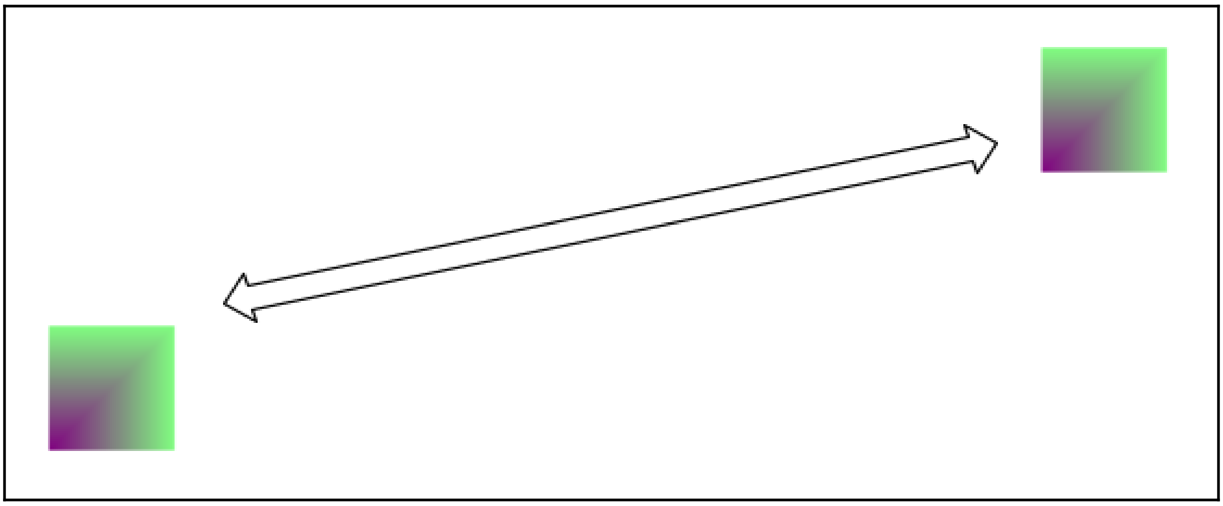
Code:

```
#include <windows.h>              // For Windows-specific OpenGL setup

#ifdef __APPLE__                  // For macOS compatibility
#include <GLUT/glut.h>
#else                             // For Windows/Linux
#include <GL/glut.h>
#endif

float y_position = 0.0;           // X-coordinate of the square's position
float speed = 0.5;                // Movement speed per frame X

//Display Function
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
   glLoadIdentity();              // Reset current transformation matrix

   glTranslated(y_position, y_position, 0.0);     // Apply X and Y translation

   //Draw Rectangle
   glBegin(GL_POLYGON);           // Start drawing a filled polygon (square)
      glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
      glVertex2f(-10, -10);      // Bottom-left vertex of the square
      glColor3f(0.5, 4.0, 0.5);
      glVertex2f(10, -10);       // Bottom-right vertex
      glColor3f(0.5, 5.0, 0.5);
      glVertex2f(10, 10);        // Top-right vertex
      glColor3f(0.5, 6.0, 0.5);
      glVertex2f(-10, 10);       // Top-left vertex
   glEnd();                      // End of polygon definition
```

```
    glutSwapBuffers();             // Swap buffers to display the current frame
}

//Reshape Function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);        // Set the viewport to match new window size

    glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
    glLoadIdentity();             // Reset projection matrix
    gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

    glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
void initOpenGL()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
{
    glutPostRedisplay();               // Mark the window to be redisplayed
    glutTimerFunc(60 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

     y_position += speed;              // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (y_position - 10 > 100)
        y_position = -100 -10;

    /*y_position -= speed;              // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (y_position - 10 <= -100)
        y_position = 100 + 10;*/
}

//Main Function
```

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);                              // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double buffering, RGBA color, and depth buffer
    glutInitWindowSize(500, 500);                       // Set window size to 500x500 pixels
    glutInitWindowPosition(100, 100);                   // Set window position on screen
    glutCreateWindow("Rectangle Re-Translate-Z");             // Create the window with an empty title

    initOpenGL();                  // Initialize OpenGL settings
    glutDisplayFunc(display);      // Register display function
    glutReshapeFunc(reshape);      // Register reshape function
    glutTimerFunc(0, timer, 0);    // Start the timer loop

    glutMainLoop();                // Enter the main event loop

    return 0;                      // Exit the program
}
```

Result: Rectangle Re-Translate-Z



Conclusions:

1. Z-axis re-translation complete.
2. Depth reversal visualized.
3. 3D motion loop achieved.

Experiment No: 14

Experiment Name: Two Rectangle Translate Rendering in OpenGL

Objectives:

1. Translate two rectangles independently.
2. Manage multiple transformations.
3. Create a dynamic layout.

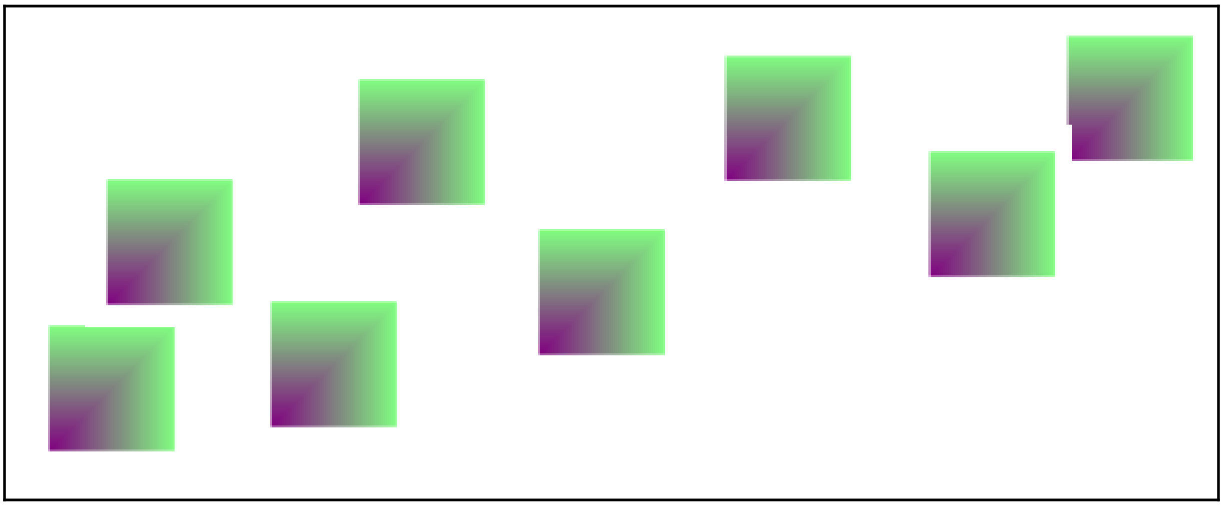
Code:

```
#include <windows.h>              // For Windows-specific OpenGL setup

#ifdef __APPLE__                  // For macOS compatibility
#include <GLUT/glut.h>
#else                     // For Windows/Linux
#include <GL/glut.h>
#endif

float x_position = 0.0, x1_position = 0.0;          // X-coordinate of the square's position
int state = 1;               // Movement speed per frame X
//Display Function
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);  // Clear the screen with background color
   glLoadIdentity();            // Reset current transformation matrix
   glTranslated(x_position, x_position, 0.0);   // Apply horizontal translation
   //Draw Rectangle
   glBegin(GL_POLYGON);           // Start drawing a filled polygon (square)
      glColor3f(0.5, 0.0, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
      glVertex2f(-10, -10);      // Bottom-left vertex of the square
      glColor3f(0.5, 4.0, 0.5);
      glVertex2f(10, -10);       // Bottom-right vertex
      glColor3f(0.5, 5.0, 0.5);
      glVertex2f(10, 10);        // Top-right vertex
      glColor3f(0.5, 6.0, 0.5);
      glVertex2f(-10, 10);       // Top-left vertex
   glEnd();

   glLoadIdentity();            // Reset current transformation matrix
   glTranslated(x1_position, 0.0, 0.0);  // Apply horizontal translation
     glBegin(GL_POLYGON);           // Start drawing a filled polygon (square)
      glColor3f(0.5, 0.4, 0.5);  // Set color to purple (R=0.5, G=0, B=0.5)
      glVertex2f(-10, -10);      // Bottom-left vertex of the square
      glColor3f(0.5, 4.0, 0.5);
      glVertex2f(10, -10);       // Bottom-right vertex
      glColor3f(0.5, 5.0, 0.5);
      glVertex2f(10, 10);        // Top-right vertex
      glColor3f(0.5, 6.0, 0.5);
      glVertex2f(-10, 10);       // Top-left vertex
   glEnd();              // End of polygon definition
                 // End of polygon definition
   glutSwapBuffers();           // Swap buffers to display the current frame
}
```

```
//Reshape Function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);        // Set the viewport to match new window size

    glMatrixMode(GL_PROJECTION);  // Switch to projection matrix
    glLoadIdentity();             // Reset projection matrix
    gluOrtho2D(-100, 100, -100, 100); // Set orthographic projection (2D view)

    glMatrixMode(GL_MODELVIEW);   // Switch back to modelview matrix
}

//Initialization Function
void initOpenGL()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);      // Set background color to black (R=0, G=0, B=0, A=0)
}

//Timer Callback Function
void timer(int)
{
    glutPostRedisplay();              // Mark the window to be redisplayed
    glutTimerFunc(60 / 60, timer, 0); // Call timer() again after ~16ms (60 FPS)

                // Move square to the right
    // If square reaches top or bottom edge, reverse direction
    if (x_position <= 110)
        x_position += 0.4;

    else
        x_position = -110;

    switch (state){

    case 1:
        if(x1_position < 85)
          x1_position += 0.4;

        else
            state =-1;
        break;

    case -1:
        if(x1_position > -85)
          x1_position -= 0.4;

        else
            state =1;
        break;

    }

    /*x_position -= speed;              // Move square to the right

    // If square reaches top or bottom edge, reverse direction
    if (x_position - 10 <= -100)
        x_position = 100 + 10;*/
}
```

```
//Main Function
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                              // Initialize GLUT
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH); // Enable double
buffering, RGBA color, and depth buffer
   glutInitWindowSize(500, 500);                       // Set window size to 500x500 pixels
   glutInitWindowPosition(100, 100);                   // Set window position on screen
   glutCreateWindow("Rectangle Re-Translate-X");              // Create the window with an
empty title

   initOpenGL();                  // Initialize OpenGL settings
   glutDisplayFunc(display);       // Register display function
   glutReshapeFunc(reshape);        // Register reshape function
   glutTimerFunc(0, timer, 0);      // Start the timer loop

   glutMainLoop();                 // Enter the main event loop

   return 0;                    // Exit the program
}
```

Result: Two Rectangle Translate



Conclusions:

1. Multi-object translation successful.
2. Scene complexity increased.
3. Independent motion verified.

Experiment No: 15

Experiment Name: Triangle and Rectangle Rendering in OpenGL

Objectives:

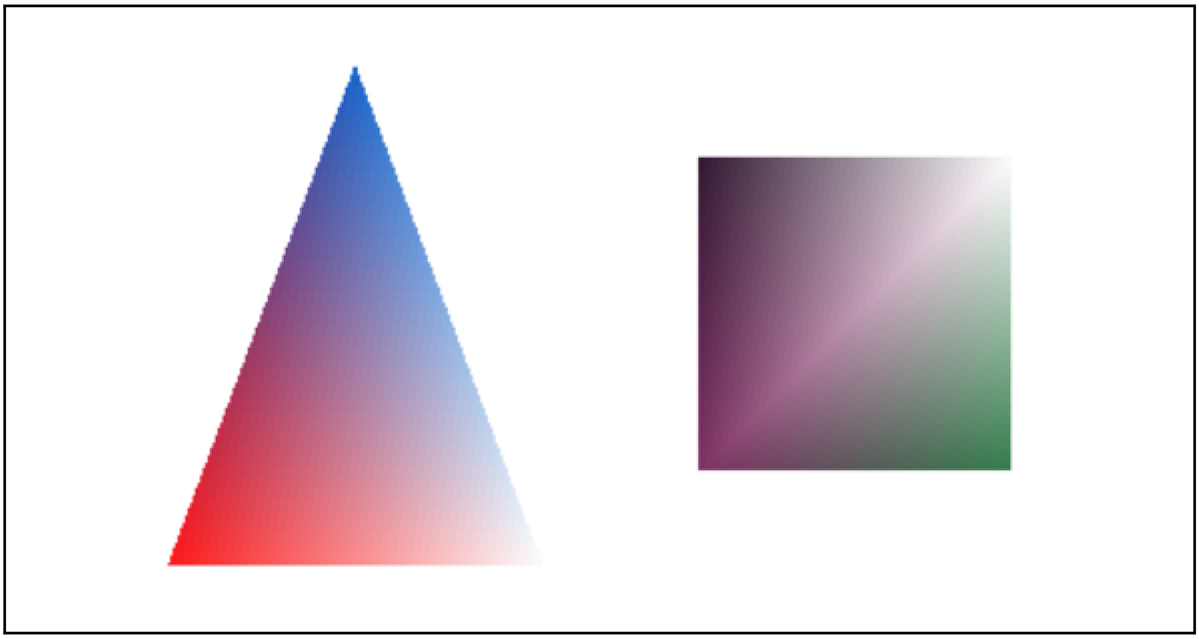
1. Render triangle and rectangle together.
2. Manage multiple shapes.
3. Explore layout composition.

Code:

```
#include <GL/gl.h>       // Include OpenGL core header
#include <GL/glut.h>     // Include GLUT header for window management

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);       // Clear the screen with the current clear color

   glBegin(GL_POLYGON);                // Begin drawing the first polygon (quadrilateral)
      glColor3f(0.5, 0.2, 0.4);       // Set current color to purple shade
      glVertex3f(25, -25, 0.0);       // Specify first vertex at (25, -25)

      glColor3f(0.20, 0.5, 0.3);      // Set color to greenish shade
      glVertex3f(75, -25, 0.0);       // Specify second vertex at (75, -25)

      glColor3f(1, 1, 1);             // Set color to white
      glVertex3f(75, 25, 0.0);        // Specify third vertex at (75, 25)

      glColor3f(0.2, 0.1, 0.2);       // Set color to dark purple/brown
      glVertex3f(25, 25, 0.0);        // Specify fourth vertex at (25, 25)
   glEnd();                           // End drawing the first polygon

   glTranslatef(-30.0, 0.0, 0.0);     // Translate the current coordinate system by -30 along X axis

   glBegin(GL_POLYGON);               // Begin drawing the second polygon (triangle)
      glColor3f(0.1, 0.4, 0.8);       // Set color to blue shade
      glVertex3f(0, 40, 0.0);         // Specify first vertex at (0, 40)

      glColor3f(1.1, 1, 1);           // Set color slightly over white (1.1 is out of normal range)
      glVertex3f(30, -40, 0.0);       // Specify second vertex at (30, -40)

      glColor3f(1, 0.1, 0.1);         // Set color to bright red shade
      glVertex3f(-30, -40, 0.0);      // Specify third vertex at (-30, -40)
   glEnd();                           // End drawing the second polygon

   glFlush();                         // Flush the rendering pipeline to display the drawn shapes
}

void init(void)
{
   glClearColor(1.0, 1.0, 1.0, 0.0);       // Set the clear (background) color to white
   glMatrixMode(GL_PROJECTION);            // Switch to projection matrix mode
   glLoadIdentity();                       // Reset the projection matrix
   glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);  // Define orthographic projection box
}
```

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                // Initialize GLUT with command line parameters
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Set display mode to single buffer and
RGB color
   glutInitWindowSize(500, 500);            // Set the initial window size to 500x500 pixels
   glutInitWindowPosition(100, 100);          // Set the initial window position on the screen
   glutCreateWindow("rgb-triangle-and-rectangle"); // Create the window with the title
"rgb-triangle-and-rectangle"
   init();                          // Call the initialization function
   glutDisplayFunc(display);                // Register display callback function
   glutMainLoop();                      // Enter the GLUT event processing loop
   return 0;                        // Return 0 to indicate successful execution
}
```

Result: Triangle and Rectangle



Conclusions:

1. Shapes rendered simultaneously.
2. Scene composition practiced.
3. Object layering understood.

Experiment No: 16

Experiment Name: Triangle with Rectangle Rendering in OpenGL

Objectives:

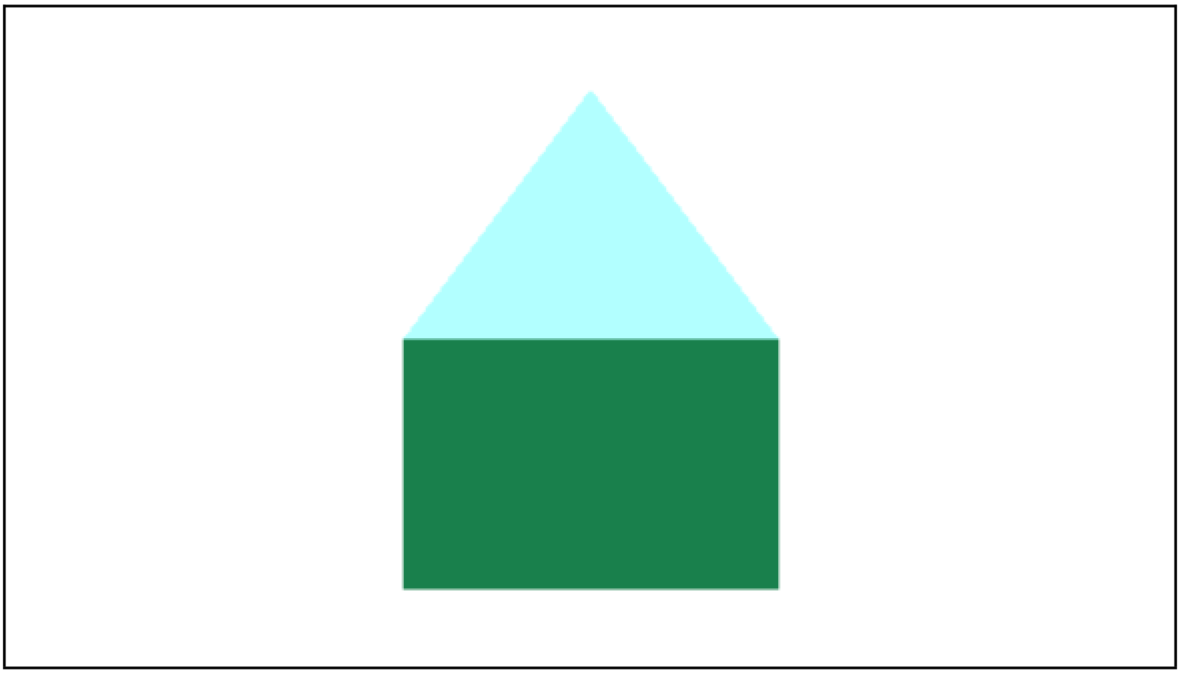
1. Combine triangle and rectangle.
2. Create a composite shape.
3. Test rendering order.

Code:

```
#include <GL/gl.h>      // Header file for OpenGL functions
#include <GL/glut.h>    // Header file for GLUT library (windowing, input, etc.)

// Function to render the display
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);        // Clear the screen with the background color

   //Draw Triangle
   glColor3f(0.7, 1.0, 1.0);           // Set color to light blue
   glBegin(GL_POLYGON);                   // Begin a polygon (triangle in this case)
      glVertex3f(-30, 0, 0.0);        // Bottom-left corner of triangle
      glVertex3f(30, 0, 0.0);         // Bottom-right corner of triangle
      glVertex3f(0, 40, 0.0);         // Top (peak) of triangle
   glEnd();                          // End the triangle polygon

   //Draw Rectangle
   glColor3f(0.1, 0.5, 0.3);           // Set color to dark green
   glBegin(GL_POLYGON);                  // Begin a polygon (rectangle)
      glVertex3f(-30, -40, 0.0);      // Bottom-left corner of rectangle
      glVertex3f(30, -40, 0.0);       // Bottom-right corner of rectangle
      glVertex3f(30, 0, 0.0);         // Top-right corner of rectangle
      glVertex3f(-30, 0, 0.0);        // Top-left corner of rectangle
   glEnd();                          // End the rectangle polygon

   glFlush();                        // Execute all drawing commands and show result
}

// Initialization function for setting background and projection
void init(void)
{
   glClearColor(1.0, 1.0, 1.0, 1.0);     // Set background color to black (R, G, B, A)
   glMatrixMode(GL_PROJECTION);         // Set current matrix mode to projection
   glLoadIdentity();                  // Reset the projection matrix
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic projection
}

// Main function - entry point of the program
int main(int argc, char** argv)
{
   glutInit(&argc, argv);                     // Initialize GLUT with command-line args
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);       // Set display mode to single buffer &
RGB color
   glutInitWindowSize(500, 500);                 // Set the window size (width x height)
```

```
    glutInitWindowPosition(100, 100);                 // Set initial position of window on screen
    glutCreateWindow("Triangle and Rectangle");    // Create window with a title

    init();                     // Call initialization function
    glutDisplayFunc(display);     // Register display callback function
    glutMainLoop();              // Enter the GLUT event-processing loop

    return 0;                   // Exit the program
}
```

Result: Triangle with Rectangle



Conclusions:

1. Composite shape formed.
2. Rendering sequence managed.
3. Shape interaction explored.

Experiment No: 17

Experiment Name: RGB Triangle and Rectangle Rendering in OpenGL

Objectives:

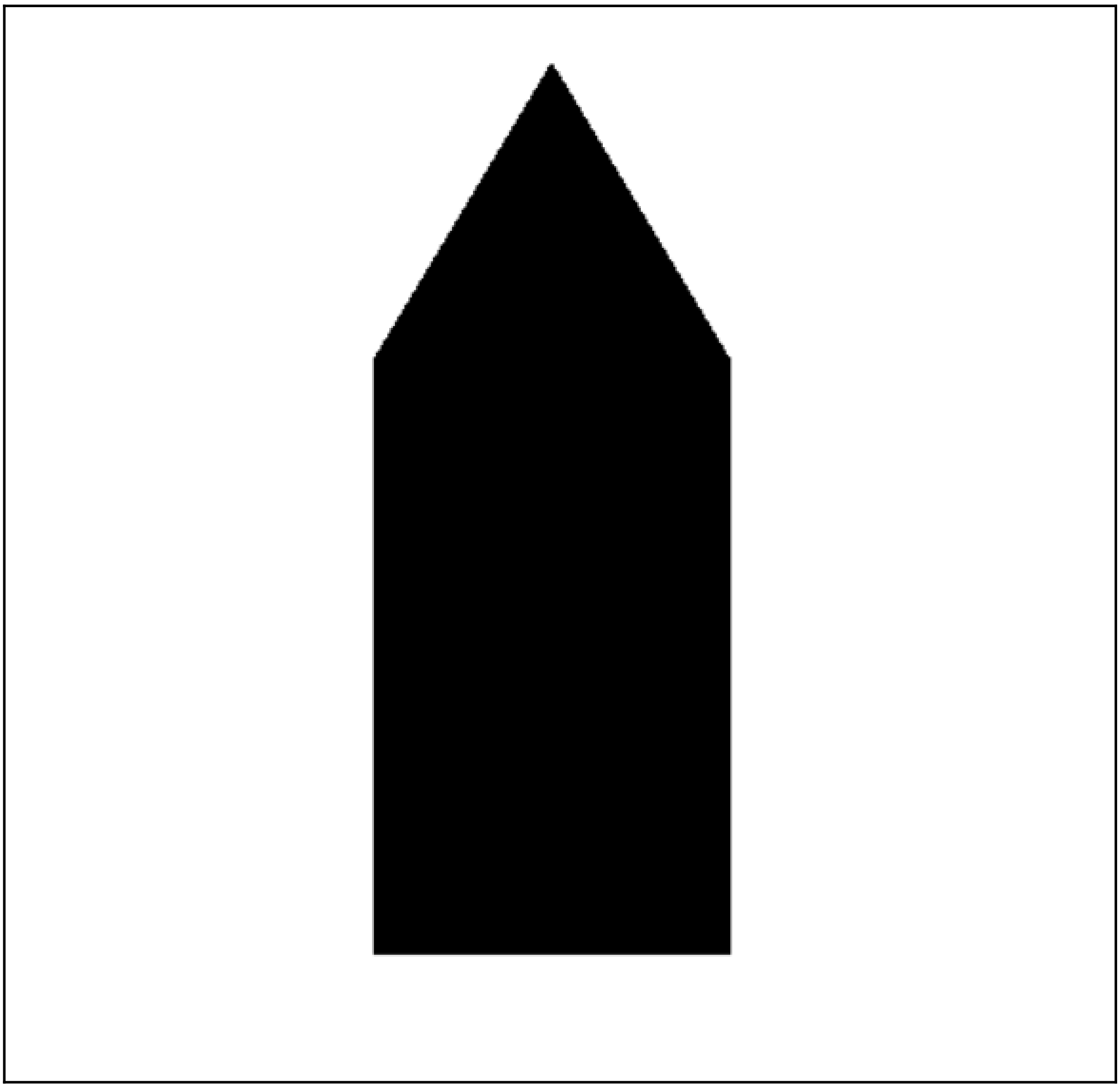
1. Apply RGB colors to both shapes.
2. Blend colors across objects.
3. Enhance visual harmony.

Code:

```
#include <GL/gl.h>          // Include OpenGL header for core functions
#include <GL/glut.h>         // Include GLUT header for windowing and event handling

void display(void)          // Display callback function to draw the scene
{
   glClear (GL_COLOR_BUFFER_BIT);      // Clear the color buffer to prepare for new frame
   glColor3f (0, 0, 0);         // Set current drawing color (white, slightly above 1.0 for intensity)

   glBegin(GL_POLYGON);               // Start defining a polygon
      glVertex3f (-30, -60, 0.0);     // Define vertex 1 of the polygon
      glVertex3f (30, -60, 0.0);     // Define vertex 2 of the polygon
      glVertex3f (30, 40, 0.0);      // Define vertex 3 of the polygon
      glVertex3f (-30, 40, 0.0);     // Define vertex 4 of the polygon
   glEnd();                         // End definition of the polygon

   glBegin(GL_POLYGON);               // Start defining another polygon (triangle)
      glVertex3f (0, 90, 0.0);       // Define vertex 1 of the polygon (top point)
      glVertex3f (30, 40, 0.0);      // Define vertex 2 of the polygon (bottom right)
      glVertex3f (-30, 40, 0.0);     // Define vertex 3 of the polygon (bottom left)
   glEnd();                         // End definition of the polygon

   glFlush ();                      // Force execution of OpenGL commands in finite time
}

void init (void)                 // Initialization function for OpenGL state setup
{
   glClearColor (1.0, 1.0, 1.0, 1.0);     // Set clear color to black with full transparency

   glMatrixMode(GL_PROJECTION);           // Switch to projection matrix mode
   glLoadIdentity();                    // Load identity matrix to reset projection
   glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Define a 2D orthographic projection volume
}

int main(int argc, char** argv)    // Main function: program entry point
{
   glutInit(&argc, argv);              // Initialize GLUT library with command line arguments
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode to single buffering
and RGB color
   glutInitWindowSize (500, 500);        // Set initial window size to 500x500 pixels
   glutInitWindowPosition (100,100);      // Set initial window position on screen
   glutCreateWindow ("triangle-with-rectangle"); // Create window with title
"triangle-with-rectangle"
   init ();                        // Call initialization function to set OpenGL states
   glutDisplayFunc(display);             // Register display callback function
```

```
    glutMainLoop();                     // Enter GLUT event processing loop
    return 0;                   // Exit program
}
```

Result: RGB Triangle and Rectangle



Conclusions:

1. Color blending is successful.
2. Multi-shape shading achieved.
3. Scene aesthetics improved.

Experiment No: 18

Experiment Name: House Rendering in OpenGL

Objectives:

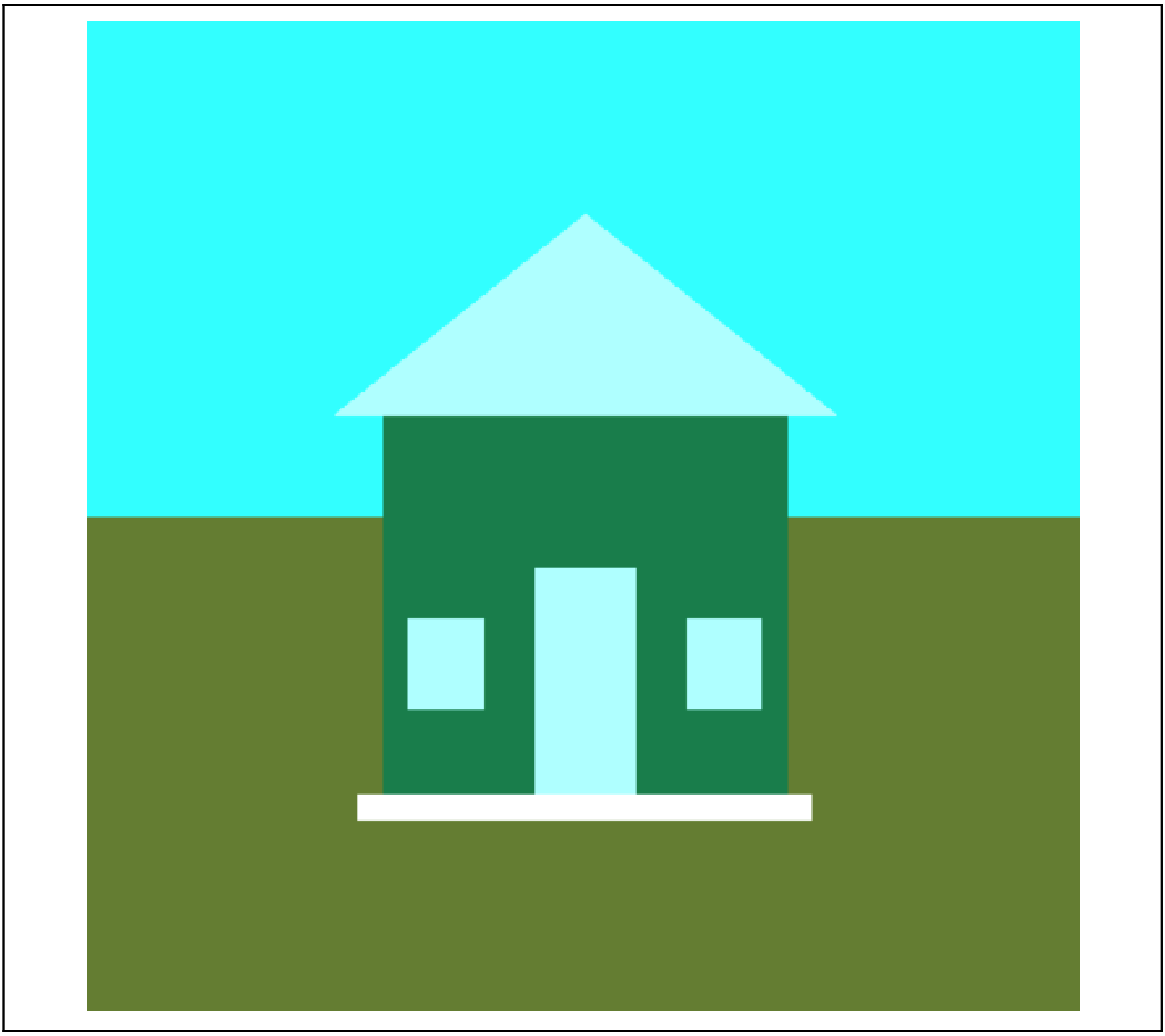
1. Construct house shape using primitives.
2. Combine triangles and rectangles.
3. Practice scene modeling.

Code:

```
#include <GL/gl.h>        // OpenGL functions for rendering
#include <GL/glut.h>       // GLUT library for windowing and event handling

// Display callback function
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT); // Clear the screen with background color

   // Draw upper background (Sky - Light Cyan)
   glColor3f(0.2, 1.0, 1.0); // Set color to light cyan
   glBegin(GL_POLYGON); // Begin drawing polygon
      glVertex3f(-100, 0, 0.0);   // Bottom-left
      glVertex3f(100, 0, 0.0);    // Bottom-right
      glVertex3f(100, 100, 0.0);  // Top-right
      glVertex3f(-100, 100, 0.0); // Top-left
   glEnd(); // End polygon

   // Draw roof (Triangle - Light Blue)
   glColor3f(0.7, 1.0, 1.0); // Set color to light blue
   glBegin(GL_POLYGON); // Begin triangle
      glVertex3f(-50, 20, 0.0);  // Bottom-left of roof
      glVertex3f(50, 20, 0.0);   // Bottom-right of roof
      glVertex3f(0, 60, 0.0);    // Top of roof (peak)
   glEnd(); // End triangle

   // Draw lower background (Ground - Olive Green)
   glColor3f(0.4, 0.5, 0.2); // Set color to olive green
   glBegin(GL_POLYGON); // Begin ground
      glVertex3f(-100, -100, 0.0); // Bottom-left
      glVertex3f(100, -100, 0.0);  // Bottom-right
      glVertex3f(100, 0, 0.0);     // Top-right
      glVertex3f(-100, 0, 0.0);    // Top-left
   glEnd(); // End ground

   // Draw house body (Dark Green)
   glColor3f(0.1, 0.5, 0.3); // Set color to dark green
   glBegin(GL_POLYGON); // Begin house rectangle
      glVertex3f(-40, -60, 0.0); // Bottom-left
      glVertex3f(40, -60, 0.0);  // Bottom-right
      glVertex3f(40, 20, 0.0);   // Top-right
      glVertex3f(-40, 20, 0.0);  // Top-left
   glEnd(); // End house body

   // Draw house footer/base (White strip)
```

```
    glColor3f(1.0, 1.0, 1.0); // Set color to white
    glBegin(GL_POLYGON); // Begin base strip
        glVertex3f(-45, -60, 0.0); // Bottom-left
        glVertex3f(45, -60, 0.0);  // Bottom-right
        glVertex3f(45, -55, 0.0);  // Top-right
        glVertex3f(-45, -55, 0.0); // Top-left
    glEnd(); // End base strip

    // Draw door (Light Blue)
    glColor3f(0.7, 1.0, 1.0); // Set color to light blue
    glBegin(GL_POLYGON); // Begin door
        glVertex3f(-10, -55, 0.0); // Bottom-left
        glVertex3f(10, -55, 0.0);  // Bottom-right
        glVertex3f(10, -10, 0.0);  // Top-right
        glVertex3f(-10, -10, 0.0); // Top-left
    glEnd(); // End door

    // Draw right window (Light Blue)
    glBegin(GL_POLYGON); // Begin right window
        glVertex3f(20, -38, 0.0);  // Bottom-left
        glVertex3f(35, -38, 0.0);  // Bottom-right
        glVertex3f(35, -20, 0.0);  // Top-right
        glVertex3f(20, -20, 0.0);  // Top-left
    glEnd(); // End right window

    // Draw left window (Light Blue)
    glBegin(GL_POLYGON); // Begin left window
        glVertex3f(-20, -38, 0.0); // Bottom-right
        glVertex3f(-35, -38, 0.0); // Bottom-left
        glVertex3f(-35, -20, 0.0); // Top-left
        glVertex3f(-20, -20, 0.0); // Top-right
    glEnd(); // End left window

    glFlush(); // Flush drawing commands and render the frame
}

// Initialization function
void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to black (R,G,B,A)

    glMatrixMode(GL_PROJECTION); // Set projection matrix
    glLoadIdentity(); // Load identity matrix (reset)
    glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Set orthographic 2D projection
}

// Main function
int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize GLUT with command-line arguments
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Set display mode: single buffer, RGB
color
    glutInitWindowSize(500, 500); // Set window size (width x height)
    glutInitWindowPosition(100, 100); // Set initial window position on screen
    glutCreateWindow("House"); // Create window with title

    init(); // Call user-defined initialization function
    glutDisplayFunc(display); // Register display callback function
    glutMainLoop(); // Enter event-processing loop (infinite loop)
    return 0; // End of main
}
```

Result: House Rendering



Conclusions:

1. House structure rendered.
2. Shape composition mastered.
3. Basic modeling skills developed.

Experiment No: 19

Experiment Name: House with Tree Rendering in OpenGL

Objectives:

1. Add a tree beside the house.
2. Expand scene complexity.
3. Practice object placement.

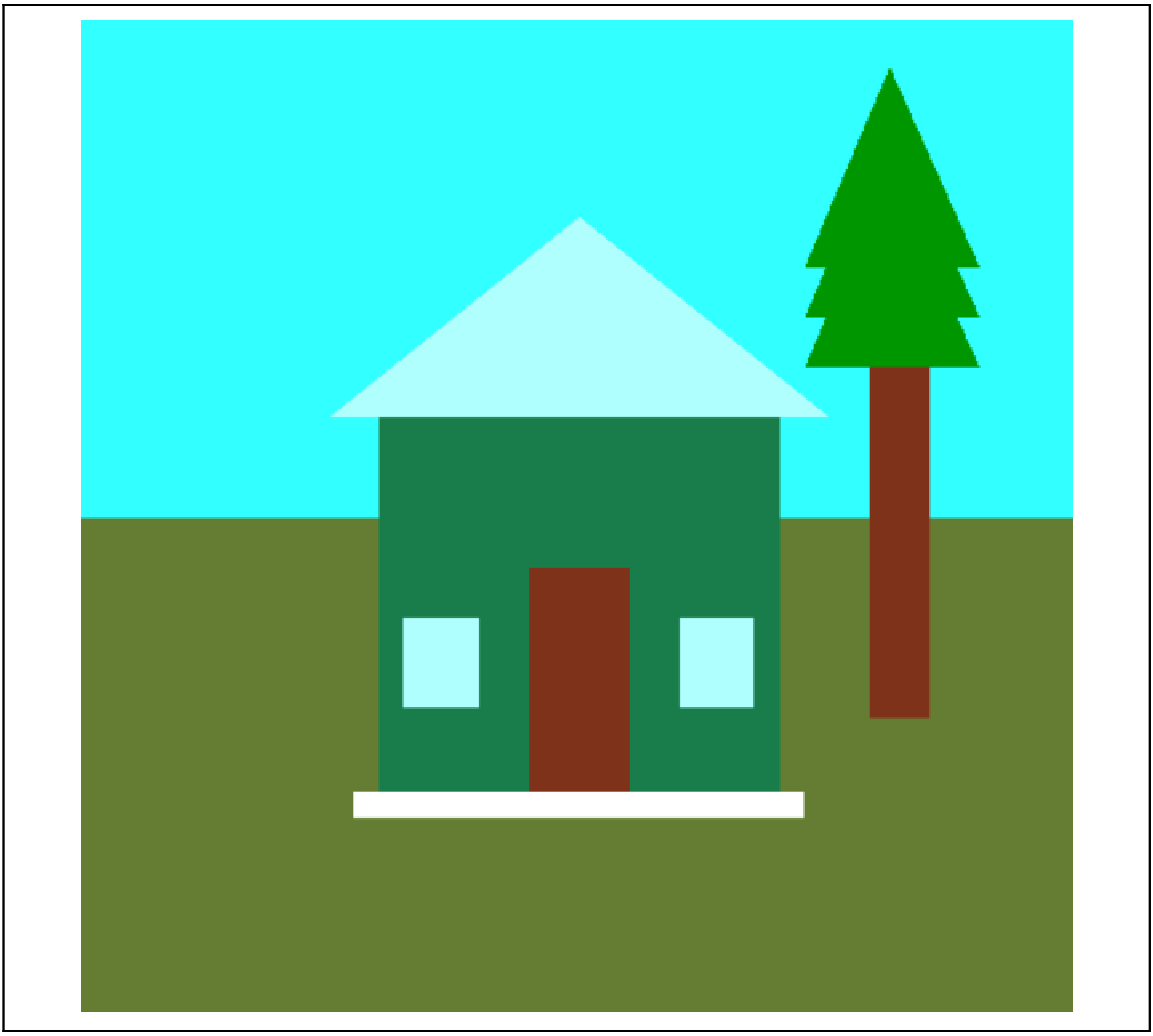
Code:

```
#include <GL/gl.h>        // OpenGL functions for rendering
#include <GL/glut.h>       // GLUT library for windowing and event handling

// Display callback function
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT); // Clear the screen with background color

   // Draw upper background (Sky - Light Cyan)
   glColor3f(0.2, 1.0, 1.0); // Set color to light cyan
   glBegin(GL_POLYGON); // Begin drawing polygon
      glVertex3f(-100, 0, 0.0);   // Bottom-left
      glVertex3f(100, 0, 0.0);    // Bottom-right
      glVertex3f(100, 100, 0.0);  // Top-right
      glVertex3f(-100, 100, 0.0); // Top-left
   glEnd(); // End polygon

   // Draw roof (Triangle - Light Blue)
   glColor3f(0.7, 1.0, 1.0); // Set color to light blue
   glBegin(GL_POLYGON); // Begin triangle
      glVertex3f(-50, 20, 0.0);  // Bottom-left of roof
      glVertex3f(50, 20, 0.0);   // Bottom-right of roof
      glVertex3f(0, 60, 0.0);    // Top of roof (peak)
   glEnd(); // End triangle

   // Draw lower background (Ground - Olive Green)
   glColor3f(0.4, 0.5, 0.2); // Set color to olive green
   glBegin(GL_POLYGON); // Begin ground
      glVertex3f(-100, -100, 0.0); // Bottom-left
      glVertex3f(100, -100, 0.0);  // Bottom-right
      glVertex3f(100, 0, 0.0);     // Top-right
      glVertex3f(-100, 0, 0.0);    // Top-left
   glEnd(); // End ground

   // Draw house body (Dark Green)
   glColor3f(0.1, 0.5, 0.3); // Set color to dark green
   glBegin(GL_POLYGON); // Begin house rectangle
      glVertex3f(-40, -60, 0.0); // Bottom-left
      glVertex3f(40, -60, 0.0);  // Bottom-right
      glVertex3f(40, 20, 0.0);   // Top-right
      glVertex3f(-40, 20, 0.0);  // Top-left
   glEnd(); // End house body

   // Draw house footer/base (White strip)
```

```
glColor3f(1.0, 1.0, 1.0); // Set color to white
glBegin(GL_POLYGON); // Begin base strip
   glVertex3f(-45, -60, 0.0); // Bottom-left
   glVertex3f(45, -60, 0.0);  // Bottom-right
   glVertex3f(45, -55, 0.0);  // Top-right
   glVertex3f(-45, -55, 0.0); // Top-left
glEnd(); // End base strip

// Draw door (Light Blue)
glColor3f(0.5, 0.2, 0.1);// Set color to dark brown
glBegin(GL_POLYGON); // Begin door
   glVertex3f(-10, -55, 0.0); // Bottom-left
   glVertex3f(10, -55, 0.0);  // Bottom-right
   glVertex3f(10, -10, 0.0);  // Top-right
   glVertex3f(-10, -10, 0.0); // Top-left
glEnd(); // End door

// Draw right window (Light Blue)
glColor3f(0.7, 1.0, 1.0);
glBegin(GL_POLYGON); // Begin right window
   glVertex3f(20, -38, 0.0);  // Bottom-left
   glVertex3f(35, -38, 0.0);  // Bottom-right
   glVertex3f(35, -20, 0.0);  // Top-right
   glVertex3f(20, -20, 0.0);  // Top-left
glEnd(); // End right window

// Draw left window (Light Blue)
glColor3f(0.7, 1.0, 1.0);
glBegin(GL_POLYGON); // Begin left window
   glVertex3f(-20, -38, 0.0); // Bottom-right
   glVertex3f(-35, -38, 0.0); // Bottom-left
   glVertex3f(-35, -20, 0.0); // Top-left
   glVertex3f(-20, -20, 0.0); // Top-right
glEnd(); // End left window

//Tree trunk
glColor3f(0.5, 0.2, 0.1);      // Dark brown
glBegin(GL_POLYGON);
   glVertex3f(70, -40, 0.0);   // Bottom-right
   glVertex3f(58, -40, 0.0);   // Bottom-left
   glVertex3f(58, 30, 0.0);    // Top-left
   glVertex3f(70, 30, 0.0);    // Top-right
glEnd();

//Tree foliage - layer 1
glColor3f(0.0, 0.6, 0.0);      // Dark green
glBegin(GL_POLYGON);
   glVertex3f(45, 30, 0.0);    // Left base
   glVertex3f(80, 30, 0.0);    // Right base
   glVertex3f(62, 70, 0.0);    // Peak
glEnd();

//Tree foliage - layer 2
glBegin(GL_POLYGON);
   glVertex3f(45, 40, 0.0);
   glVertex3f(80, 40, 0.0);
   glVertex3f(62, 80, 0.0);
glEnd();
```

```
    //Tree foliage - layer 3
    glBegin(GL_POLYGON);
        glVertex3f(45, 50, 0.0);
        glVertex3f(80, 50, 0.0);
        glVertex3f(62, 90, 0.0);
    glEnd();

    glFlush(); // Flush drawing commands and render the frame
}

// Initialization function
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0); // Set background color to black (R,G,B,A)

    glMatrixMode(GL_PROJECTION); // Set projection matrix
    glLoadIdentity(); // Load identity matrix (reset)
    glOrtho(-100, 100, -100, 100, -1.0, 1.0); // Set orthographic 2D projection
}

// Main function
int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize GLUT with command-line arguments
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Set display mode: single buffer, RGB
color
    glutInitWindowSize(500, 500); // Set window size (width x height)
    glutInitWindowPosition(100, 100); // Set initial window position on screen
    glutCreateWindow("House with Tree"); // Create window with title

    init(); // Call user-defined initialization function
    glutDisplayFunc(display); // Register display callback function
    glutMainLoop(); // Enter event-processing loop (infinite loop)

    return 0; // End of main
}
```

Result: House with Tree



Conclusions:

1. Tree and house rendered together.
2. Scene realism improved.
3. Layout design enhanced.

Experiment No: 20

Experiment Name: Final Lab Project (Sayed) Rendering in OpenGL

Objectives:

1. Integrate all learned techniques.
2. Create a complete scene.
3. Demonstrate OpenGL proficiency.

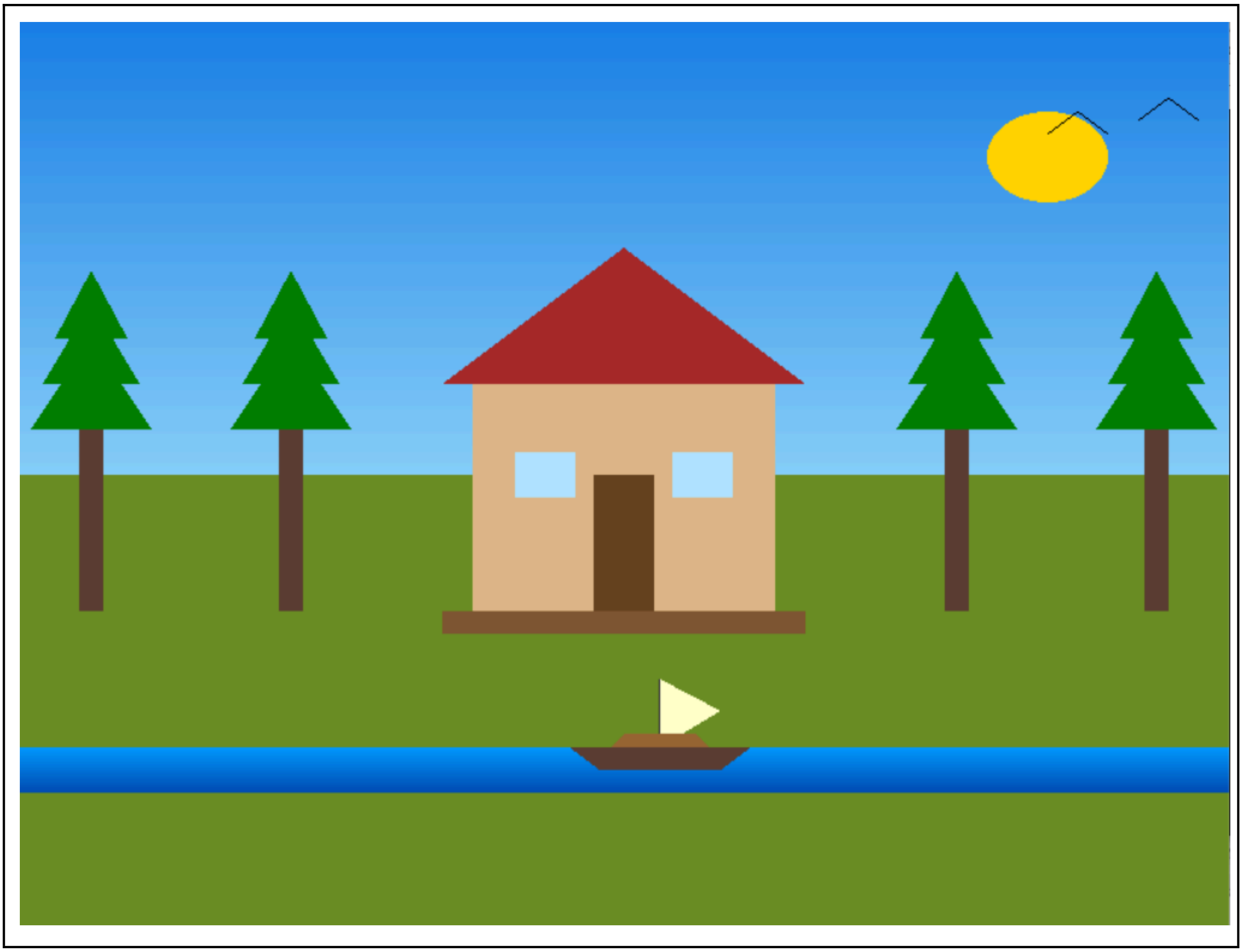
Code:

```
#include <GL/gl.h>                // Include OpenGL functions
#include <GL/glut.h>               // Include GLUT library for windowing and event handling
#include <math.h>                 // Include math functions (cos, sin, etc.)
#define PI 3.1416               // Define the value of PI for angle calculations

float boatx = 0.0;              // Variable to track boat's horizontal position
float birdx = 0.0;              // Variable to track birds' horizontal position

// Function to draw the sky with a vertical gradient
void sky() {
   glBegin(GL_QUADS);            // Begin drawing a quadrilateral
   glColor3f(0.53, 0.81, 0.98);    // Set bottom color of sky to light blue
   glVertex2f(-100, 0);          // Bottom-left corner of sky
   glVertex2f(100, 0);          // Bottom-right corner of sky

   glColor3f(0.1, 0.5, 0.9);       // Set top color of sky to deeper blue
   glVertex2f(100, 100);          // Top-right corner of sky
   glVertex2f(-100, 100);          // Top-left corner of sky
   glEnd();                    // End drawing quadrilateral
}

// Function to draw the ground
void ground() {
   glColor3f(0.42, 0.56, 0.14);    // Set ground color (greenish)
   glBegin(GL_QUADS);             // Begin drawing a quadrilateral
   glVertex2f(-100, -100);        // Bottom-left of ground
   glVertex2f(100, -100);         // Bottom-right of ground
   glVertex2f(100, 0);           // Top-right of ground
   glVertex2f(-100, 0);          // Top-left of ground
   glEnd();                    // End drawing
}

// Function to draw the sun using triangle fan (circle)
void sun(float cx, float cy, float r) {
   glColor3f(1.0, 0.84, 0.0);       // Set sun color to yellow
   glBegin(GL_TRIANGLE_FAN);      // Begin triangle fan for circular shape
   glVertex2f(cx, cy);           // Center of the sun
   for (int i = 0; i <= 100; i++) {
      float angle = 2 * PI * i / 100;              // Calculate angle
      glVertex2f(cx + r * cos(angle), cy + r * sin(angle)); // Calculate and set vertex on
circumference
   }
   glEnd();                    // End drawing
}
```

```
// Function to draw a bird using line strip
void bird(float x, float y) {
   glColor3f(0, 0, 0);           // Set color to black
   glBegin(GL_LINE_STRIP);       // Begin drawing connected lines
   glVertex2f(x, y);             // First point
   glVertex2f(x + 5, y + 5);     // Middle upward point
   glVertex2f(x + 10, y);        // Ending point
   glEnd();                      // End drawing
}

// Function to draw a tree
void tree(float x) {
   glPushMatrix();               // Save current transformation
   glTranslatef(x, 0, 0);        // Translate tree horizontally

   glColor3f(0.36, 0.25, 0.20);  // Set trunk color (brown)
   glBegin(GL_QUADS);            // Begin drawing trunk
   glVertex2f(-2, -30);          // Bottom-left of trunk
   glVertex2f(2, -30);           // Bottom-right of trunk
   glVertex2f(2, 10);            // Top-right of trunk
   glVertex2f(-2, 10);           // Top-left of trunk
   glEnd();                      // End drawing trunk

   glColor3f(0, 0.5, 0);         // Set leaf color (green)
   glBegin(GL_TRIANGLES);        // Begin first leaf layer
   glVertex2f(-10, 10);          // Left point
   glVertex2f(10, 10);           // Right point
   glVertex2f(0, 30);            // Top point
   glEnd();                      // End triangle

   glBegin(GL_TRIANGLES);        // Begin second leaf layer
   glVertex2f(-8, 20);
   glVertex2f(8, 20);
   glVertex2f(0, 38);
   glEnd();                      // End triangle

   glBegin(GL_TRIANGLES);        // Begin third leaf layer
   glVertex2f(-6, 30);
   glVertex2f(6, 30);
   glVertex2f(0, 45);
   glEnd();                      // End triangle

   glPopMatrix();                // Restore transformation
}

// Function to draw a house with roof, walls, doors and windows
void house(float x) {
   glPushMatrix();               // Save transformation
   glTranslatef(x, 0, 0);        // Translate house horizontally

   glColor3f(0.65, 0.16, 0.16);  // Set roof color (dark red)
   glBegin(GL_TRIANGLES);        // Begin roof triangle
   glVertex2f(-10, 20);
   glVertex2f(50, 20);
   glVertex2f(20, 50);
   glEnd();                      // End triangle

   glColor3f(0.87, 0.72, 0.53);  // Set house body color (beige)
```

```
    glBegin(GL_QUADS);          // Begin drawing house body
    glVertex2f(-5, -30);
    glVertex2f(45, -30);
    glVertex2f(45, 20);
    glVertex2f(-5, 20);
    glEnd();                    // End house body

    glColor3f(0.4, 0.26, 0.13);    // Set door color (brown)
    glBegin(GL_QUADS);          // Draw door
    glVertex2f(15, -30);
    glVertex2f(25, -30);
    glVertex2f(25, 0);
    glVertex2f(15, 0);
    glEnd();                    // End door

    glColor3f(0.7, 0.9, 1.0);      // Set window color (light blue)
    glBegin(GL_QUADS);          // Draw left window
    glVertex2f(2, -5);
    glVertex2f(12, -5);
    glVertex2f(12, 5);
    glVertex2f(2, 5);
    glEnd();                    // End left window

    glBegin(GL_QUADS);          // Draw right window
    glVertex2f(28, -5);
    glVertex2f(38, -5);
    glVertex2f(38, 5);
    glVertex2f(28, 5);
    glEnd();                    // End right window

    glColor3f(0.5f, 0.35f, 0.2f);  // Set veranda floor color
    glBegin(GL_QUADS);          // Draw veranda floor
    glVertex2f(-10, -30);
    glVertex2f(50, -30);
    glVertex2f(50, -35);
    glVertex2f(-10, -35);
    glEnd();                    // End veranda

    glPopMatrix();              // Restore transformation
}

// Function to draw a river with color gradient
void river() {
    glBegin(GL_QUADS);          // Begin river
    glColor3f(0.0, 0.3, 0.7);      // Bottom color (dark blue)
    glVertex2f(-1000, -70);
    glVertex2f(1000, -70);

    glColor3f(0.0, 0.6, 1.0);      // Top color (light blue)
    glVertex2f(900, -60);
    glVertex2f(-900, -60);
    glEnd();                    // End river
}

// Function to draw a moving boat
void boat(float x) {
    float y = -60;              // Set vertical position of boat
    glPushMatrix();             // Save transformation
    glTranslatef(x, y, 0);      // Move boat to position (x, y)
    glColor3f(0.36, 0.25, 0.20);  // Set base color
    glBegin(GL_QUADS);            // Draw base of the boat
    glVertex2f(-15, 0);
    glVertex2f(15, 0);
    glVertex2f(10, -5);
    glVertex2f(-10, -5);
    glEnd();                      // End base

    glColor3f(0.2, 0.2, 0.2);     // Set mast color
    glBegin(GL_LINES);            // Draw mast
    glVertex2f(0, 0);
    glVertex2f(0, 15);
    glEnd();                      // End mast

    glColor3f(1, 1, 0.8);         // Set sail color
    glBegin(GL_TRIANGLES);        // Draw sail
    glVertex2f(0, 15);
    glVertex2f(0, 0);
    glVertex2f(10, 8);
    glEnd();                      // End sail

    glColor3f(0.6, 0.4, 0.2);     // Set cabin color
    glBegin(GL_QUADS);            // Draw cabin
    glVertex2f(-8, 0);
    glVertex2f(8, 0);
    glVertex2f(6, 3);
    glVertex2f(-6, 3);
    glEnd();                      // End cabin

    glPopMatrix();                // Restore transformation
}

// Wrapper function to draw sun at a fixed location
void sun() {
    sun(70, 70, 10);              // Call sun drawing with center (70,70) and radius 10
}

// Wrapper function to draw multiple birds
void birds() {
    bird(birdx, 75);              // First bird
    bird(birdx + 15, 78);         // Second bird
    bird(birdx + 35, 78);         // Third bird
    bird(birdx + 50, 75);         // Fourth bird
}

// Wrapper function to draw animated boat
void boat() {
    boat(boatx - 30);             // Draw boat at shifted x position
}

// Wrapper function to draw house
void house() {
    house(-20);                   // Draw house at x = -20
}

// Wrapper function to draw multiple trees
void trees() {
    tree(-55);                    // Draw tree at x = -55
    tree(-88);                  // Draw tree at x = -88
    tree(55);                   // Draw tree at x = 55
    tree(88);                   // Draw tree at x = 88
}

// Timer callback function for animation
void timer(int value) {
    boatx += 1.0;               // Increment boat x-position
    birdx += 2.0;               // Increment bird x-position

    if (boatx > 150)            // Reset boat if out of screen
        boatx = -150;
    if (birdx > 150)            // Reset bird if out of screen
        birdx = -150;

    glutPostRedisplay();        // Request screen redraw
    glutTimerFunc(30, timer, 0); // Call timer again after 30 milliseconds
}

// Main display function
void display() {
    glClear(GL_COLOR_BUFFER_BIT); // Clear the screen

    sky();                      // Draw sky
    ground();                   // Draw ground
    river();                    // Draw river
    sun();                      // Draw sun
    birds();                    // Draw birds
    boat();                     // Draw boat
    house();                    // Draw house
    trees();                    // Draw trees

    glFlush();                  // Flush OpenGL commands
}

// OpenGL initialization
void init() {
    glClearColor(0, 0, 0, 0);     // Set background color to black
    glMatrixMode(GL_PROJECTION); // Set projection matrix mode
    glLoadIdentity();            // Reset projection matrix
    glOrtho(-100, 100, -100, 100, -1, 1); // Set orthographic projection
}

// Main function - entry point
int main(int argc, char** argv) {
    glutInit(&argc, argv);                          // Initialize GLUT
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);        // Set display mode to single buffer
and RGB
    glutInitWindowSize(800, 600);                   // Set window size
    glutInitWindowPosition(100, 100);               // Set window position
    glutCreateWindow("cse-336-project");            // Create window with title

    init();                                     // Call initialization function
    glutDisplayFunc(display);                       // Set display function callback
    glutTimerFunc(0, timer, 0);                     // Set timer function for animation
    glutMainLoop();                             // Enter the main event loop
    return 0;                                   // Exit program
}
```

Result: Final Lab Project (Sayed)



Conclusions:

1. Final project completed.
2. All rendering concepts applied.
3. Strong grasp of OpenGL confirmed.

End!