NOTRE DAME UNIVERSITY BANGLADESH

Computer Graphics Lab Report-03

Course Code: CSE-4204
Course Title: Computer Graphics Lab
Lab Topic: Usage of Loop
Lab Task: Draw Emoji

Submitted by:

Name: Istiak Alam
ID: 0692230005101005
Batch: CSE-20

Submission Date: February 6, 2026

Submitted to:

Humayara Binte Rashid
Lecturer, Dept. of CSE
Notre Dame University Bangladesh

Table of Contents

1 Lab Work : Usage of Loop

1.1 Objective

The objective of this lab is to learn how to use loops in OpenGL with GLUT to create repetitive 2D graphical patterns efficiently. In this experiment, a chessboard is drawn by applying nested loops to systematically generate alternating black and white squares. Through this exercise, students gain practical experience in combining loops with OpenGL drawing functions, managing coordinates programmatically, and understanding how iteration can simplify the creation of structured graphical objects in a 2D orthographic projection.

1.2 Practice Code

The program demonstrates the use of nested loops in OpenGL with GLUT to generate a structured 2D pattern, specifically an 8x8 chessboard. The `init()` function sets up the OpenGL environment by defining a clear background color and an orthographic projection suitable for 2D drawing. The `display()` function contains nested `for` loops. The outer loop iterates through the rows of the chessboard, while the inner loop iterates through the columns. Conditional logic is applied within the loops to alternate the square colors between black and white. The `glRectf()` function is used to draw each individual square at the calculated coordinates, with the positions updated programmatically in each iteration. The `main()` function initializes GLUT, sets up the display window, registers the `display()` callback function, and enters the GLUT event loop. This structure highlights how iteration and coordinate management can simplify the creation of repetitive graphical elements in OpenGL.

```
/*
 * GLUT ChessBoard using Loop
 *
 * Written by Istiak Alam
 *
 */

#include <GL/glut.h>  // Include GLUT header here
#include <stdlib.h>
#include <math.h>

// Initialization
void init()
{
    glClearColor(0.0f, 0.8f, 1.0f, 0.0f); // Background color
    glOrtho(-100, 100, -100, 100, -10, 10);  // Set up an orthogonal
        view
}

void display(void)
{
```

```
    glClear(GL_COLOR_BUFFER_BIT);

    //int x1=-100, y1=60, x2=-90, y2=70;
    int startX = -50;
    int startY = 60;
    int size = 10;

    for (int r = 1; r<=8; r++)
    {
        int x1 = startX;
        int x2 = -40;

        for (int c = 1; c <=8; c++)
        {
            if ((r + c) % 2 == 0)
                glColor3f(1.0f, 1.0f, 1.0f); // white
            else
                glColor3f(0.0f, 0.0f, 0.0f); // black

            glRectf(x1, startY, x2, startY - size);

            x1=x1+size;
            x2=x2+size;
        }

        startY=startY-size;
    }

    glFlush();
}

// 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
    glutInitWindowSize(700, 700);  // Set window size
    glutInitWindowPosition(650, 200);  // Set window position
    glutCreateWindow("Chess Board");  // Create the window

    init();  // Initialize OpenGL settings
    glutDisplayFunc(display);  // Register the display function

    glutMainLoop();  // Enter the GLUT event loop
    return 0;
}
```

1.3 Output

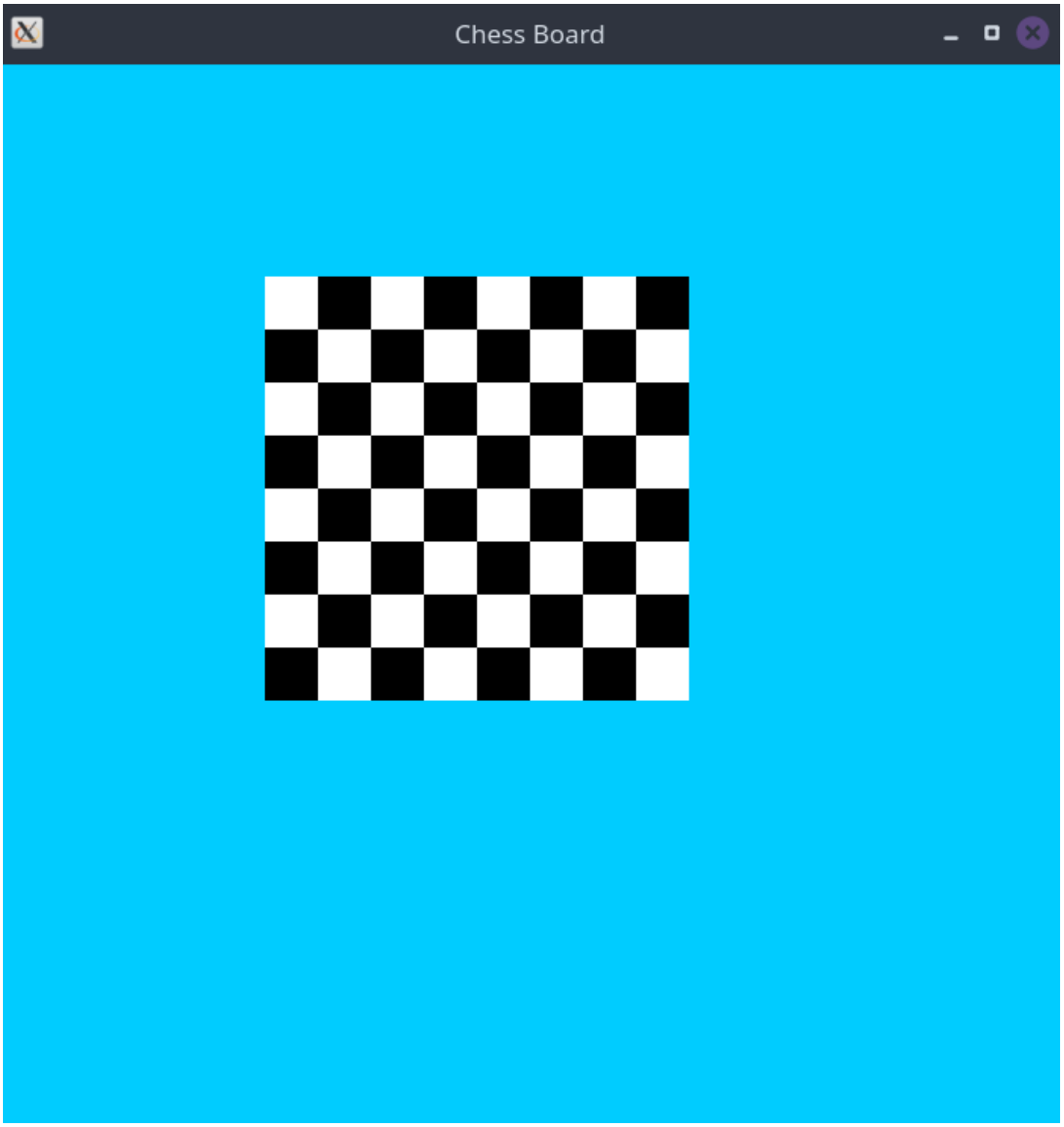

Figure 1: chess Board Using OpenGL

2 Lab Task : Neutral Face Emoji

2.1 Objective

The objective of this lab is to understand and apply fundamental 2D computer graphics concepts using OpenGL and GLUT by drawing a Neutral Face Emoji. The experiment focuses on constructing an emoji using basic geometric primitives such as circles and rectangles, along with iterative techniques for curve approximation.
Through this task, students gain hands-on experience with coordinate systems, color specification, custom shape creation using trigonometric functions, and rendering 2D objects within an orthographic projection.

2.2 Tools and Environment

- Programming Language: C++
- Graphics Library: OpenGL with GLUT
- IDE: Code::Blocks
- OS: Kali Linux

2.3 Source Code

The program illustrates the construction of a 2D emoji using OpenGL primitives and custom drawing functions. The OpenGL environment is initialized with a white background and an orthographic projection suitable for 2D rendering.

A custom function, `DrawCircle()`, is implemented using a loop and trigonometric calculations to approximate a filled circular shape, which forms the main face of the emoji. Another function, `DrawCircleOutline()`, draws the circular boundary using a line loop for visual clarity.
The `display()` function controls the rendering process by drawing the face, eyes, and mouth using combinations of circles and rectangles. Color values are assigned using `glColor3f()` to achieve a realistic emoji appearance.

The `main()` function initializes GLUT, creates the display window, registers the display callback function, and starts the rendering loop. This structure demonstrates the use of loops, geometric primitives, and coordinate manipulation in OpenGL.

```c
/*
 * Neutral Face Emoji Drawing
 *
 * Written by Istiak Alam
 *
 */

#include <GL/glut.h>  // GLUT header
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265358979323846


// Draw filled circle
void DrawCircle(float cx, float cy, float rx, float ry, int segments
    )
{
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(cx, cy); // center point
    for (int i = 0; i <= segments; i++)
    {
        float theta = 2.0f * PI * i / segments;
        float x = rx * cos(theta);
        float y = ry * sin(theta);
        glVertex2f(cx + x, cy + y);
    }
    glEnd();
}

// Draw circle outline
void DrawCircleOutline(float cx, float cy, float r, int segments)
{
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < segments; i++)
    {
        float theta = 2.0f * PI * i / segments;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();
}

void init()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50, 50, -50, 50, -1, 1);

    glMatrixMode(GL_MODELVIEW);
```

```
}

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

    /* Face (filled) */
    glColor3f(1.0f, 0.85f, 0.0f);          // Emoji yellow
    DrawCircle(0, 0, 25, 25, 100);

    /* Face outline */
    glColor3f(0.0f, 0.0f, 0.0f);
    glLineWidth(3);
    DrawCircleOutline(0, 0, 25, 100);


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

    // Left eye
    glRectf(-16.0f, 6.5f, -6.0f, 4.5f);

    // Right eye
    glRectf(6.0f, 6.5f, 16.0f, 4.5f);

    // Mouth
    glRectf(-10.0f, -6.0f, 10.0f, -8.0f);

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize (800, 800);
        glutInitWindowPosition (500, 100);
        glutCreateWindow ("Emoji ");
        init();
    glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

2.4 Output

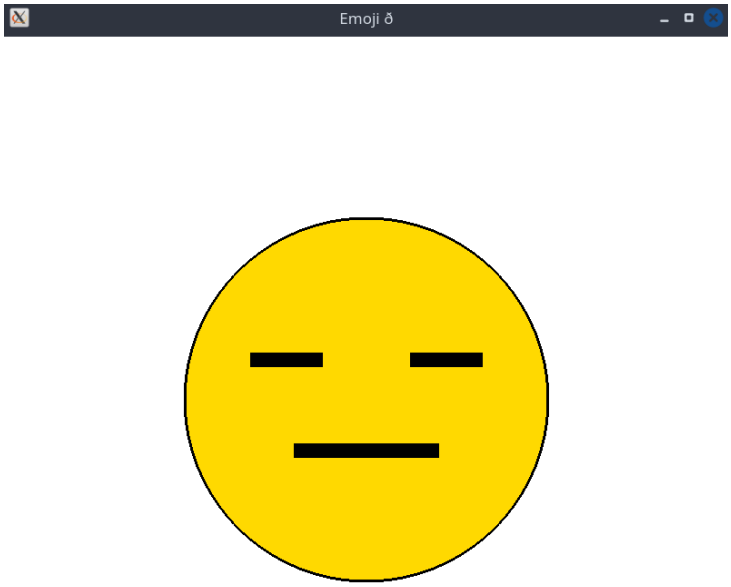

Figure 2: Neutral Face Emoji Using OpenGL

2.5 Discussion

This experiment demonstrates how basic OpenGL drawing primitives and user-defined functions can be combined to create a recognizable 2D graphical object. The use of loops with trigonometric functions highlights how smooth curves, such as circles, can be approximated in computer graphics. The lab also reinforces the concept of orthographic projection, which simplifies object positioning and scaling in a 2D coordinate system. Proper use of color selection and geometric alignment contributed to the accurate visual representation of the neutral face emoji. Overall, the experiment provides practical insight into procedural drawing and shape composition in OpenGL.

2.6 Conclusion

In this lab experiment, a Neutral Face Emoji was successfully drawn using OpenGL and GLUT. The task strengthened understanding of 2D coordinate systems, geometric primitives, loop-based curve generation, and color specification. This experiment serves as a solid foundation for advanced computer graphics topics such as transformations, animation, and interactive graphics, and enhances problem-solving skills in graphical programming using OpenGL.

3 Lab Task : Palestine Flag Emoji

3.1 Objective

The objective of this lab is to practice fundamental 2D graphics concepts using OpenGL and GLUT. The experiment focuses on drawing a graphical representation of the Palestinian flag emoji by combining basic geometric primitives such as quadrilaterals and triangles along with custom wave-like curves. Through this task, students will gain hands-on experience with coordinate systems, color manipulation, custom shape generation, and rendering techniques in a 2D orthographic projection.

3.2 Tools and Environment

- Programming Language: C++
- Graphics Library: OpenGL with GLUT
- IDE: Code::Blocks
- OS: Kali Linux

3.3 Graph Implementation

Here is the Graph file Link

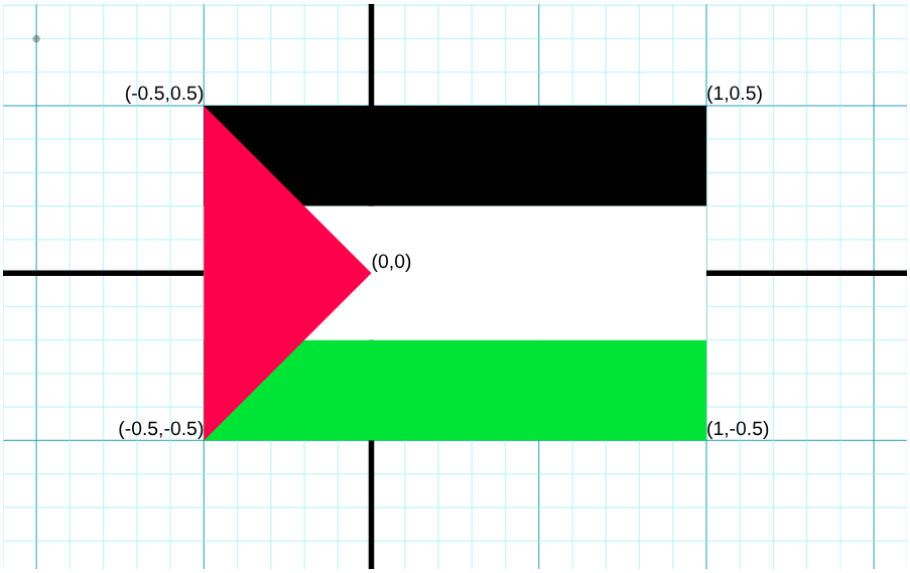

Figure 3: Graph View

3.4 Source Code

The program demonstrates how to construct a complex 2D graphical object using OpenGL primitives. It begins by initializing the OpenGL environment with a white background and defining a 2D orthographic projection. A custom function, `DrawWavyStripe()`, is

implemented to generate the flag's stripes with a smooth wavy effect using sine functions. The `display()` function orchestrates the rendering process by drawing the flag's colored stripes and the red triangle on the left side to complete the flag. Colors are set using `glColor3f()`, and geometric primitives such as `GL_QUADS` and `GL_TRIANGLES` are used to construct the shapes. The `main()` function initializes GLUT, creates the window, registers the display callback, and starts the rendering loop. The approach highlights the application of basic geometry, color assignment, and procedural drawing techniques in OpenGL.

```
/* Lab Task-03
 * Flag Emoji of Palestine
 * Drawn using OpenGL (GLUT)
 *
 * Author: Istiak Alam
 */

#include <GL/glut.h>
#include <math.h>

#define PI 3.14159265358979323846

// Function to draw wavy rectangle stripe
void DrawWavyStripe(float xStart, float xEnd, float yStart, float
   yEnd, float waveAmplitude, float waveFrequency)
{
    int segments = 100; // more segments = smoother curve
    glBegin(GL_QUADS);
    for (int i = 0; i < segments; i++)
    {
        float t1 = (float)i / segments;
        float t2 = (float)(i + 1) / segments;

        float x1 = xStart + t1 * (xEnd - xStart);
        float x2 = xStart + t2 * (xEnd - xStart);

        // Apply simple sine wave for top and bottom edges
        float yTop1 = yEnd + waveAmplitude * sin(waveFrequency * x1
           * PI);
        float yTop2 = yEnd + waveAmplitude * sin(waveFrequency * x2
           * PI);

        float yBot1 = yStart + waveAmplitude * sin(waveFrequency *
           x1 * PI);
        float yBot2 = yStart + waveAmplitude * sin(waveFrequency *
           x2 * PI);

        glVertex2f(x1, yBot1);
        glVertex2f(x2, yBot2);
        glVertex2f(x2, yTop2);
```

```
        glVertex2f(x1, yTop1);
    }
    glEnd();
}

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

    // Draw the flag stripes
    float waveAmplitude = 0.03f;  // How curved the stripes are
    float waveFrequency = 2.0f;   // Number of waves

    // Top stripe (Black)
    glColor3f(0.0f, 0.0f, 0.0f);
    DrawWavyStripe(-0.5f, 1.0f, 0.17f, 0.5f, waveAmplitude,
       waveFrequency);

    // Middle stripe (White)
    glColor3f(1.0f, 1.0f, 1.0f);
    DrawWavyStripe(-0.5f, 1.0f, -0.17f, 0.17f, waveAmplitude,
       waveFrequency);

    // Bottom stripe (Green)
    glColor3f(0.0f, 0.6f, 0.0f);
    DrawWavyStripe(-0.5f, 1.0f, -0.5f, -0.17f, waveAmplitude,
       waveFrequency);

    // Left red triangle
    glColor3f(0.8f, 0.0f, 0.0f);
    glBegin(GL_TRIANGLES);
    glVertex2f(-0.5f, -0.5f);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(-0.5f, 0.5f);
    glEnd();

    glFlush();
}

void init()
{
    glClearColor(1, 1, 1, 1); // white background
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.2, 1.2, -0.7, 0.7, -1, 1);
    glMatrixMode(GL_MODELVIEW);
}

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

```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(700, 400);
    glutInitWindowPosition(600, 200);
    glutCreateWindow("Free-Palestine");

    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

3.5 Output

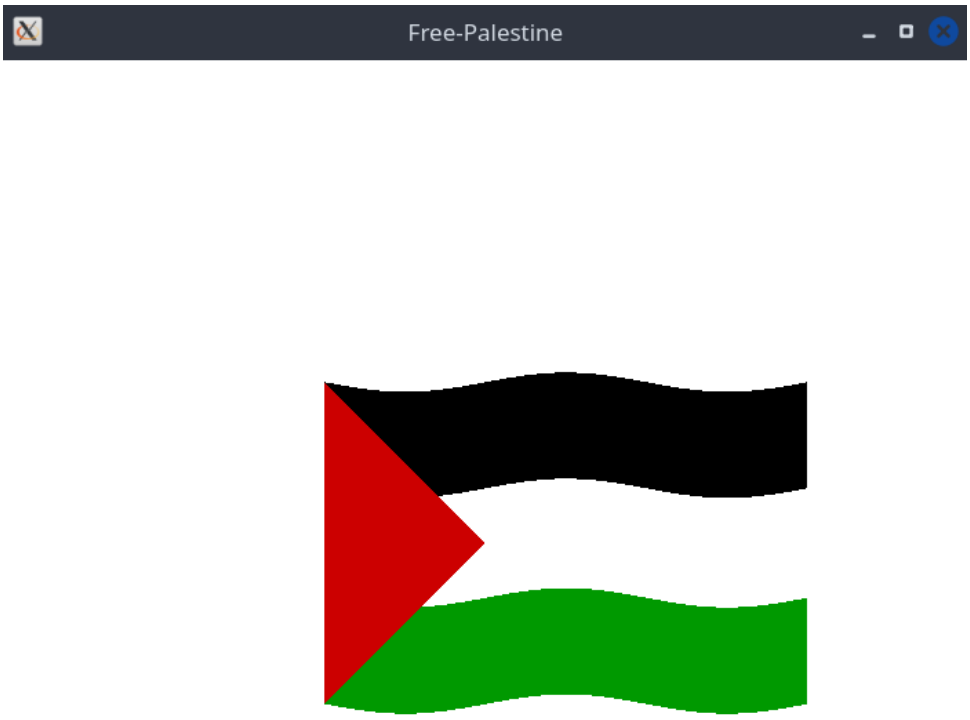

Figure 4: Output of Palestine Flag Using OpenGL

3.6 Discussion

This experiment demonstrates how elementary OpenGL primitives and custom functions can be combined to create a visually meaningful 2D object. Implementing the wavy stripes using trigonometric functions shows how curves can be approximated in a discrete graphical environment.

The lab also reinforces understanding of orthographic projection for accurate positioning and scaling in 2D space. Additionally, color blending and structured drawing sequences were applied to achieve the authentic appearance of the Palestinian flag emoji. Overall, the experiment provides practical experience in procedural graphics, geometric transformations, and visual composition in OpenGL.

3.7 Conclusion

The lab successfully achieved the construction of a 2D Palestinian flag emoji using OpenGL and GLUT. Through this experiment, key concepts such as 2D coordinate systems, geometric primitives, color specification, and custom curve generation were effectively applied.

This exercise strengthens foundational skills in graphical programming and prepares students for more advanced topics such as transformations, animation, and interactive graphics development in OpenGL.