Exercise List #3

Victor F. Ferrari - RA 187890
vferrari@mpc.com.br
MO814A/MC937A - Topics in Computer Graphics
Universidade Estadual de Campinas

March 30, 2020

1 Basic OpenGL

Consider that you have a function `listOfVertices()` that generates a list of properly formatted vertices in OpenGL. Write the OpenGL code needed to draw a red polygon composed of these vertices.
For the following code to work, the function `listOfVertices()` returns a float vector, containing the vertices in the correct format.

```
#include <iostream>
#include <vector>

#include <GL/glew.h>
#include <GLFW/glfw3.h>


int main(){

    // Initialize GLFW
    if( !glfwInit() ){
        std::cerr<<"Failed to initialize GLFW\n"<<std::endl;
        return -1;
    }

    // Open a window and create its OpenGL context
    GLFWwindow* window = glfwCreateWindow(600, 600, "Red Polygon", nullptr,nullptr);

    // Check if window was successfully created.
    if( window == nullptr ){
        std::cerr<<"Failed to open GLFW window"<<std::endl;
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent(window);

    // Initialize GLEW
    if (glewInit() != GLEW_OK){
        std::cerr<<"Failed to initialize GLEW"<<std::endl;
```

```
        glfwTerminate();
        return -1;
    }

    // Get Polygon
    std::vector<float> polygon = listOfVertices();

    GLuint vbo;
    GLuint vao = 0;

    // VBO
    // Generate and bind buffers (1)
    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);

    // Initialize buffer data.
    glBufferData(GL_ARRAY_BUFFER, polygon.size() * sizeof(float), polygon.data(),
                 GL_STATIC_DRAW);

    // VAO
    // Generate and bind vertex arrays (1).
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    // Enable vertex attribute array.
    glEnableVertexAttribArray(0);

    // Define an array of generic vertex attribute data.
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, nullptr);

    // Polygon Color : RED.
    glColor3f(1.f, 0.f, 0.f);

    while(glfwWindowShouldClose(window) == 0 ){

        // Draw polygon from the currently bound VAO.
        glDrawArrays(GL_POLYGON, 0, polygon.size());

        // Display
        glfwSwapBuffers(window);

        // Update other events (close button)
        glfwPollEvents();
    }

    // Close OpenGL window and terminate GLFW
    glfwTerminate();

    return 0;
}
```

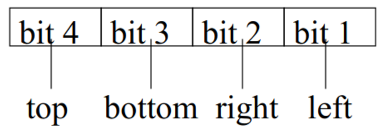


Figure 1: Cohen-Sutherland 2D Codes

2 Cohen-Sutherland Line Clipping

For each of the following line segments,

1. **Determine the 2D Cohen-Sutherland codes for each vertex.**
2. **Determine whether this line is trivially rejected, trivially accepted, or clipped.**
3. **If the line segment must be clipped, compute the new endpoints.**

NOTE: Viewport size is 800x600, origin is at (0, 0) (bottom-left corner), points are defined in window coordinates.
In the Cohen-Sutherland algorithm, each bit position indicates whether the point is inside or outside of a specific window edge. Each bit indicates one edge, as can be seen in Figure 1.

- **Line Segment 1: (-200, 700), (400, -300)**

c_0: 1001, c_1: 0100.
Both vertices are outside the window, but $c_0 \wedge c_1 = 0$, so this line is not trivially rejected. Clipping is necessary. The line slope is $m = \frac{700-(-300)}{-200-400} = -\frac{5}{3}$, so the line equation for the segment is $3y = -5x + 1100$. The intersection with the top line, $y = 600$, is at $x = -140$, so the new line segment is (-140, 600), (400, -300). Repeating the process...

c_0: 0001, c_1: 0100.
Both vertices are outside the window, but $c_0 \wedge c_1 = 0$, so this line is not trivially rejected. Clipping is necessary. The intersection with the left line, $x = 0$, is at $y = \frac{1100}{3} \approx 367$, so the new line segment is (0, 367), (400, -300). Repeating the process...

c_0: 0000, c_1: 0100.
One vertex is inside the window, and the other is outside the window, so this line is not trivially rejected or accepted ($c_0 \wedge c_1 = 0$). Clipping is necessary. The intersection with the bottom line, $y = 0$, is at $x = 200$, so the new line segment is (0, 367), (200, 0). Repeating the process...

c_0: 0000, c_1: 0000.
Both vertices are inside the window, with code 0, so $c_0 \vee c_1 = 0$. Therefore, this line is trivially accepted, and no clipping is necessary. Final segment: (0, 367), (200, 0).

- **Line Segment 2: (100, 100), (400, 600)**

c_0: 0000, c_1: 0000.
Both vertices are inside the window, with code 0, so $c_0 \vee c_1 = 0$. Therefore, this line is trivially accepted, and no clipping is necessary. Final segment: (100, 100), (400, 600).

- **Line Segment 3: (400, 300), (1000, 300)**

c_0: 0000, c_1: 0010.
One vertex is inside the window, and the other is outside the window, so this line is not trivially rejected or accepted ($c_0 \wedge c_1 = 0$). Clipping is necessary. The line slope is $m = \frac{300-300}{400-1000} = 0$, so the line equation for the segment is $y = 300$. The intersection with the right line, $x = 800$, is at $y = 300$, so the new line segment is (400, 300), (800, 300). Repeating the process...

c_0: 0000, c_1: 0000.
Both vertices are inside the window, with code 0, so $c_0 \vee c_1 = 0$. Therefore, this line is trivially accepted, and no clipping is necessary. Final segment: (400, 300), (800, 300).

3 Polygon Clipping

1. **Clip the polygon in figure 2 using the Sutherland-Hodgman algorithm against the left, right, top, and bottom sides (in that order). NOTE: Just draw the output (approximately, but indicate vertices), don't need to show any math.**

2. **Clip the polygon in figure 2 using the Weiler-Atherton algorithm. NOTE: Just draw the output (approximately, but indicate vertices), don't need to show any math.**

The output of the Sutherland-Hodgman is one polygon, but has edges on top of the viewport, while the output of the Weiler-Atherton algorithm is composed of 3 different polygons. Every new vertex introduced has a different letter, following alphabetic order, from A to Z. If a letter is missing from the image, that vertex was introduced during the execution, and later discarded. The outputs can be seen in Figure 3.
The output of Sutherland-Hodgman is in figure 3a and the output of Weiler-Atherton is in figure 3b. The main difference, apart from the vertex labels, are the edges marked by the blue arrows.

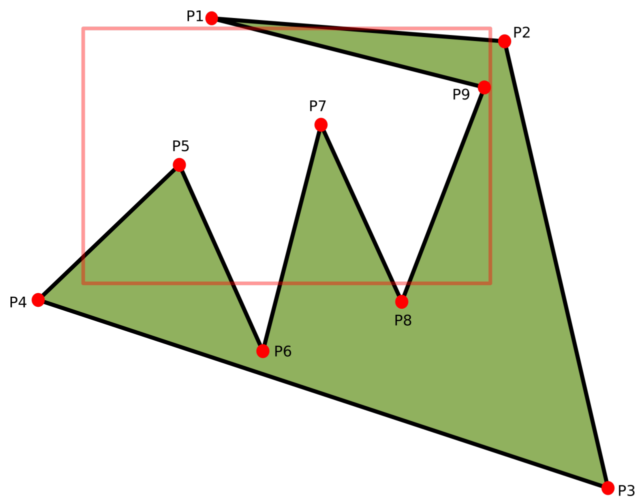
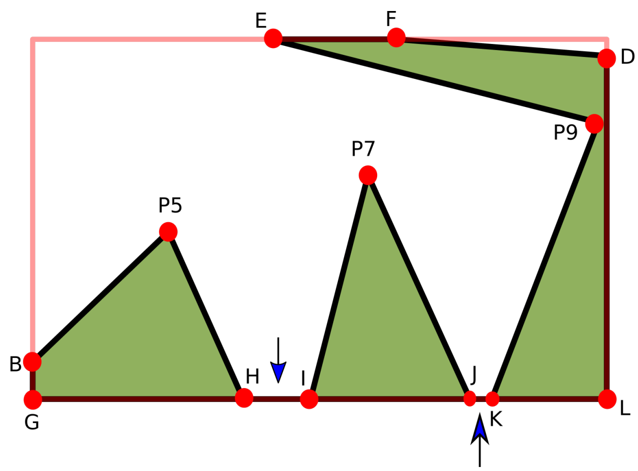
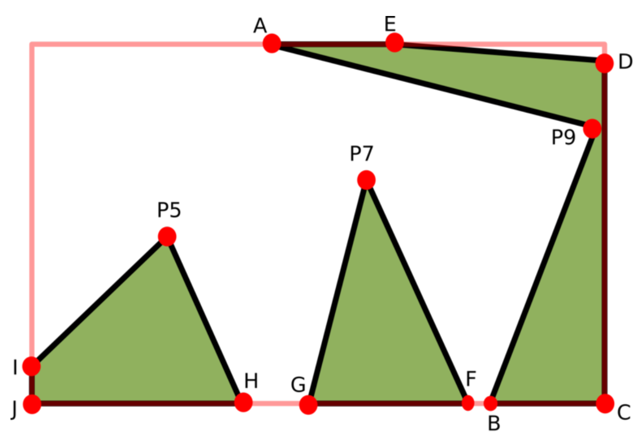


Figure 2: Polygon for clipping.



(a) Sutherland-Hodgman



(b) Weiler-Atherton

Figure 3: Outputs of clipping the polygon.