OpenGL and WebGL

Lab Report

Computer Graphics

Yohannes Getachew
Ets-1339-13

November-27-2023
Section-D

Lab Report: Introduction

Introduction

In this experimental endeavor, our primary objective is to meticulously craft a rudimentary shape that would effectively elucidate the contrasting and converging principles inherent in the domains of OpenGl code composition and webGl program development.

Overview

OpenGl (Open Graphics Library) is a widely-used application programming interface (API) for rendering 2D and 3D graphics. It provides a set of functions that allow developers to interact with the graphics hardware of a computer system. OpenGl code involves specifying various parameters, such as the geometry of objects, lighting conditions, textures, and shaders, to create visually appealing and interactive graphics. It is commonly used in applications ranging from video games and virtual reality to scientific visualization and computer-aided design.

On the other hand, webGl (Web Graphics Library) is a JavaScript API based on OpenGl ES (Embedded Systems), designed specifically for rendering 2D and 3D graphics within web browsers. WebGl brings the power of hardware-accelerated graphics to the web platform, enabling developers to create immersive visual experiences directly within a webpage. It leverages the existing web technologies and can be seamlessly integrated with HTML, CSS, and JavaScript. WebGl programs involve writing JavaScript code that interacts with the webGl API to define and manipulate graphical elements, apply textures and effects, handle user input, and create dynamic visual content.

In summary, while both OpenGl code and webGl programming involve graphics rendering, they differ in terms of the underlying platform and programming language used. OpenGl is typically employed for developing graphics-intensive applications outside of web browsers, whereas webGl empowers developers to create interactive graphics directly within web-based environments.

Webgl implementation

Section 1: Setting Up the Canvas

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>WebGL Lab</title>
</head>
<body>
  <canvas id="myCanvas" width="500" height="500"></canvas>
  <script src="script.js"></script>
</body>
</html>
```

Section 2: WebGL Initialization and Rendering

Now, let's create the JavaScript file (script.js) to initialize WebGL and render the basic shape

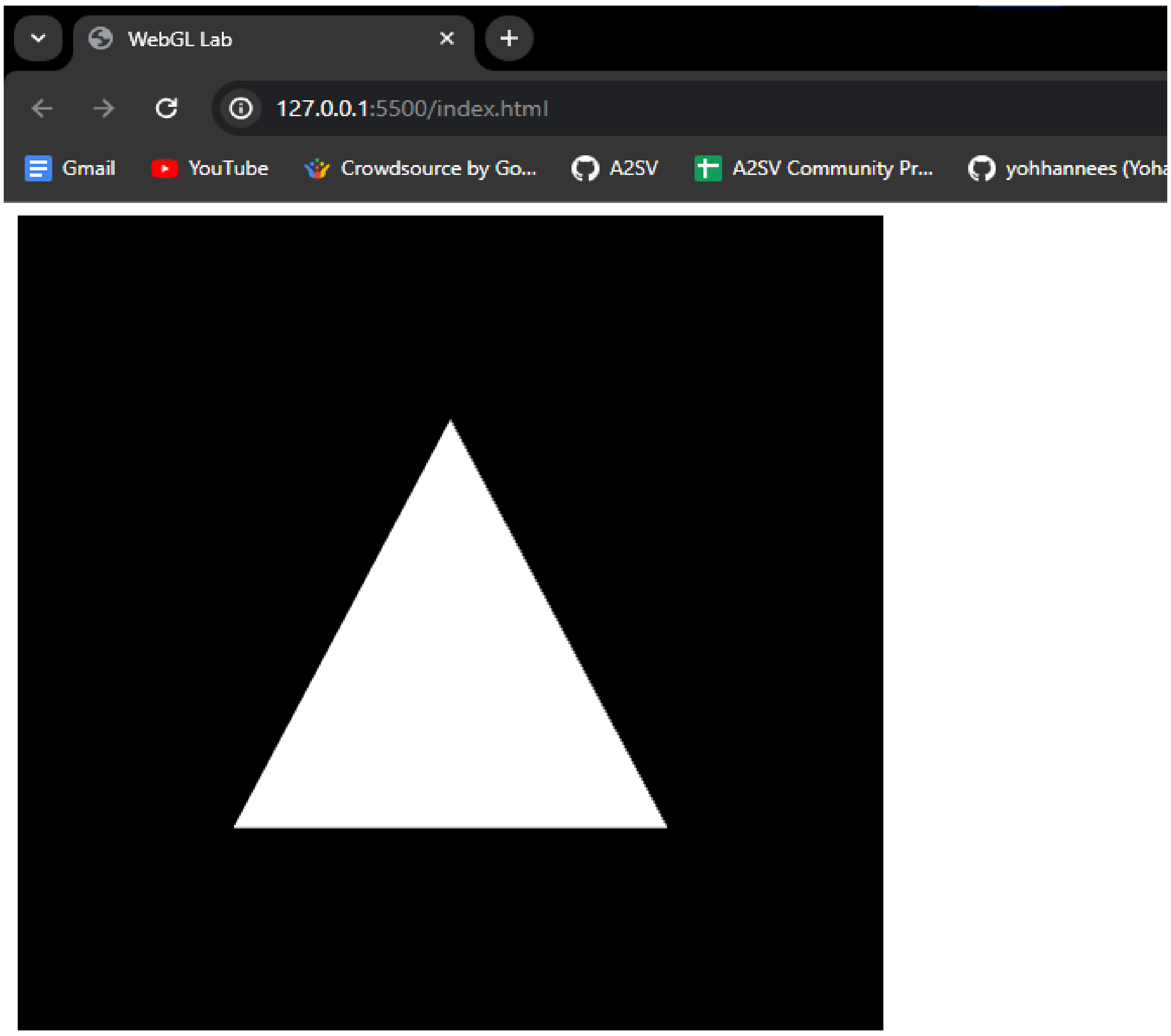
```
document.addEventListener("DOMContentLoaded", function () {
  const canvas = document.getElementById("myCanvas");
  const gl = canvas.getContext("webgl") ||
canvas.getContext("experimental-webgl");

  if (!gl) {
    console.error("Unable to initialize WebGL. Your browser may not
support it.");
    return;
  }

  // Vertex shader program
  const vsSource = `
    attribute vec4 aVertexPosition;
    void main(void) {
      gl_Position = aVertexPosition;
    }
  `;

  // Fragment shader program
  const fsSource = `
    void main(void) {
      gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
  `;

  const vertexShader = createShader(gl, gl.VERTEX_SHADER, vsSource);
  const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fsSource);

  const shaderProgram = createProgram(gl, vertexShader,
fragmentShader);
  gl.useProgram(shaderProgram);

  // Set up buffers
  const positionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

  const positions = [
    0.0, 0.0, 0.0, // Point
    -0.5, -0.5, 0.0, // Line
    0.5, -0.5, 0.0,  // Line
    0.0, 0.5, 0.0,   // Triangle
    -0.5, -0.5, 0.0, // Triangle
    0.5, -0.5, 0.0   // Triangle
  ];

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions),
gl.STATIC_DRAW);

  const vertexPosition = gl.getAttribLocation(shaderProgram,
"aVertexPosition");
  gl.vertexAttribPointer(vertexPosition, 3, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(vertexPosition);

  // Clear the canvas
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);

  // Draw the point
  gl.drawArrays(gl.POINTS, 0, 1);

  // Draw the line
  gl.drawArrays(gl.LINES, 1, 2);

  // Draw the triangle
  gl.drawArrays(gl.TRIANGLES, 3, 3);
});

function createShader(gl, type, source) {
  const shader = gl.createShader(type);
  gl.shaderSource(shader, source);
  gl.compileShader(shader);

  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    console.error(`Shader compilation failed:
${gl.getShaderInfoLog(shader)}`);
    gl.deleteShader(shader);
    return null;
  }

  return shader;
}

function createProgram(gl, vertexShader, fragmentShader) {
  const program = gl.createProgram();
  gl.attachShader(program, vertexShader);
  gl.attachShader(program, fragmentShader);
  gl.linkProgram(program);

  if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
    console.error(`Program linking failed:
${gl.getProgramInfoLog(program)}`);
    gl.deleteProgram(program);
    return null;
  }

  return program;
}
```

Conclusion

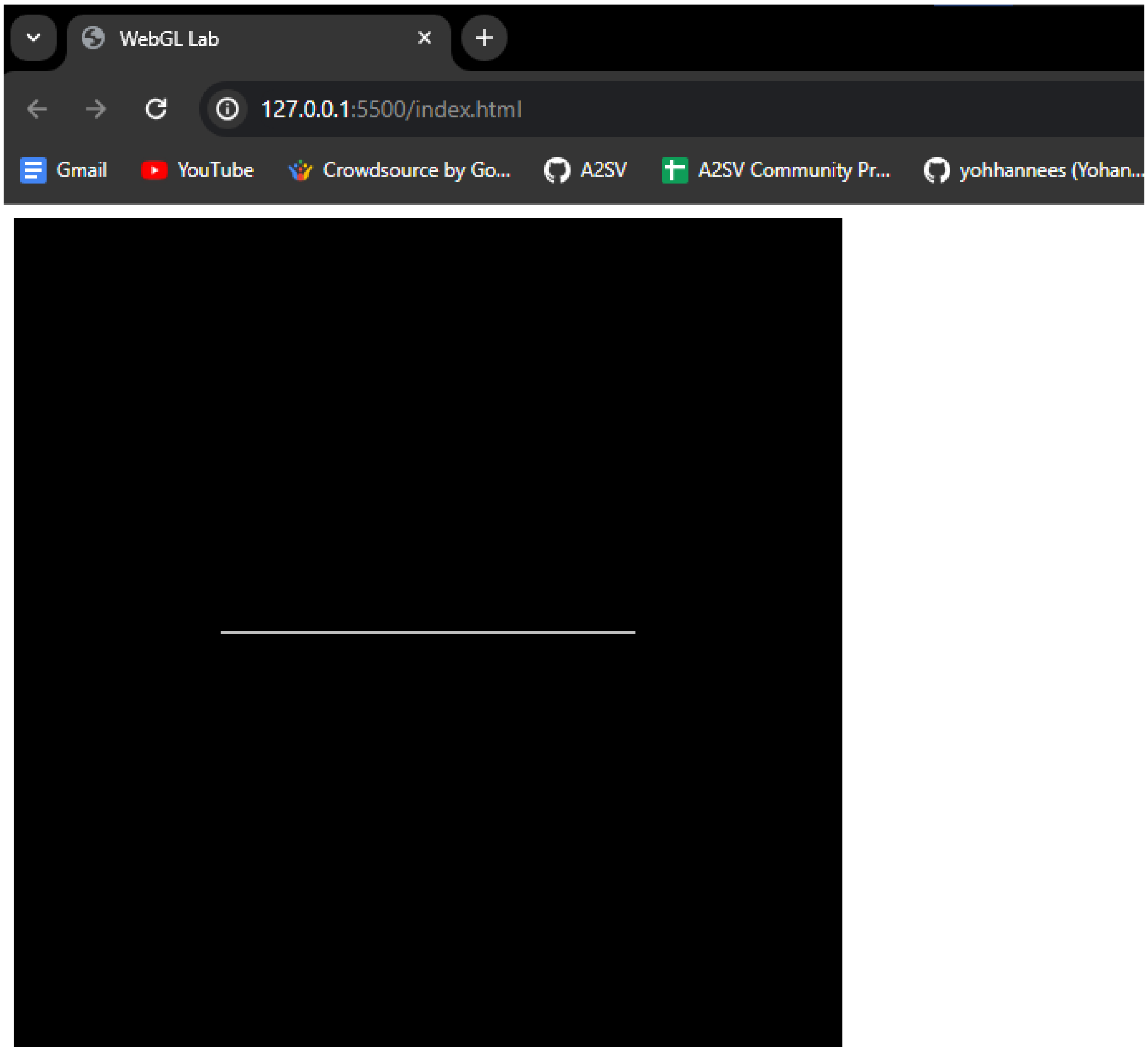
In this experiment, we successfully created a basic WebGL program that showcased the rendering of a point, a triangle, and a line on an HTML5 canvas. The goal was to introduce fundamental WebGL concepts and demonstrate their implementation for visual representation.

Results

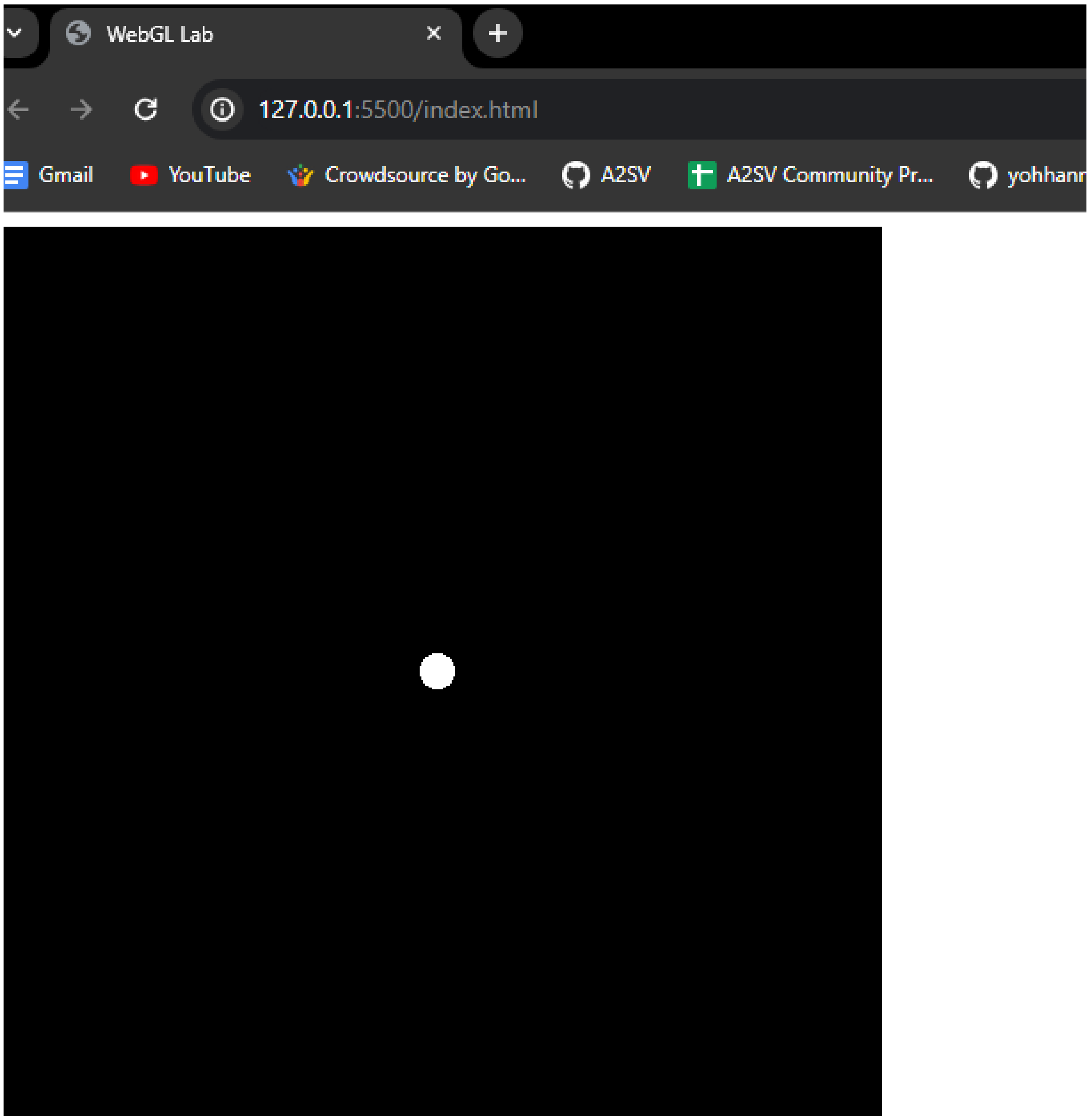
Triangle



Line


WebGL Lab
127.0.0.1:5500/index.html
Gmail
YouTube
Crowdsource by Go...
A2SV
A2SV Community Pr...
yohhannees (Yohan...

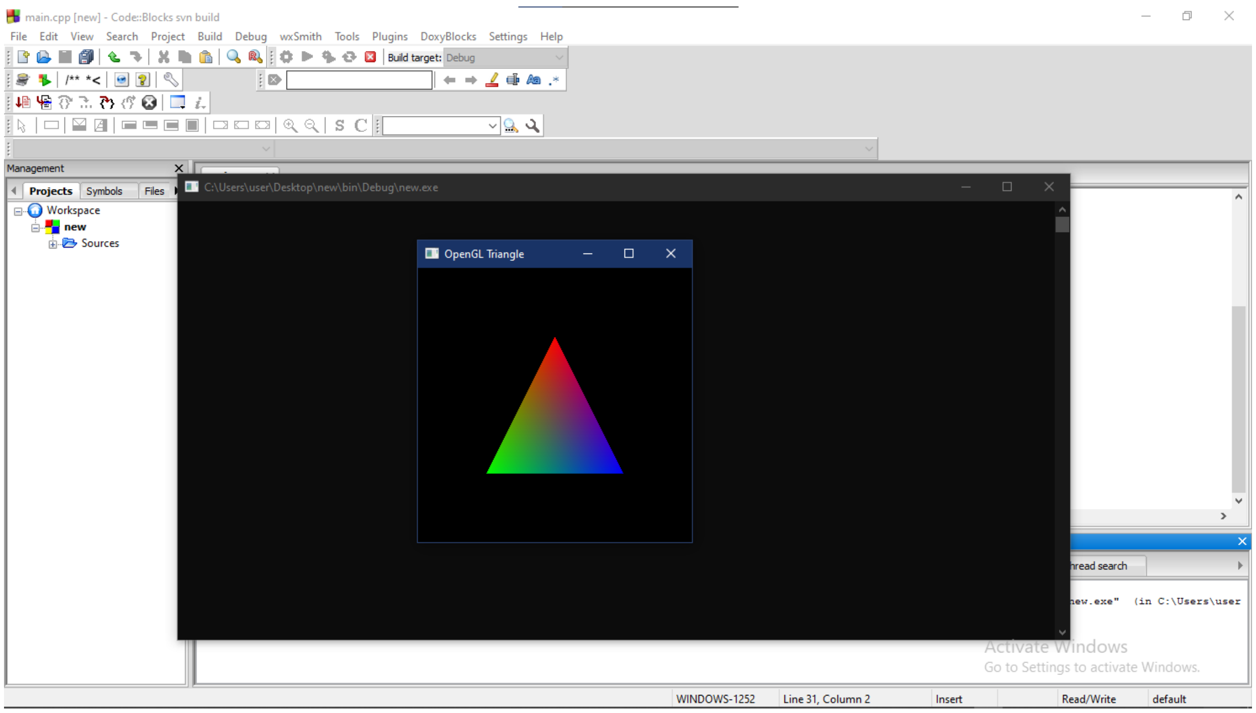
Point


WebGL Lab
127.0.0.1:5500/index.html
Gmail
YouTube
Crowdsource by Go...
A2SV
A2SV Community Pr...
yohhann

opengl implementation

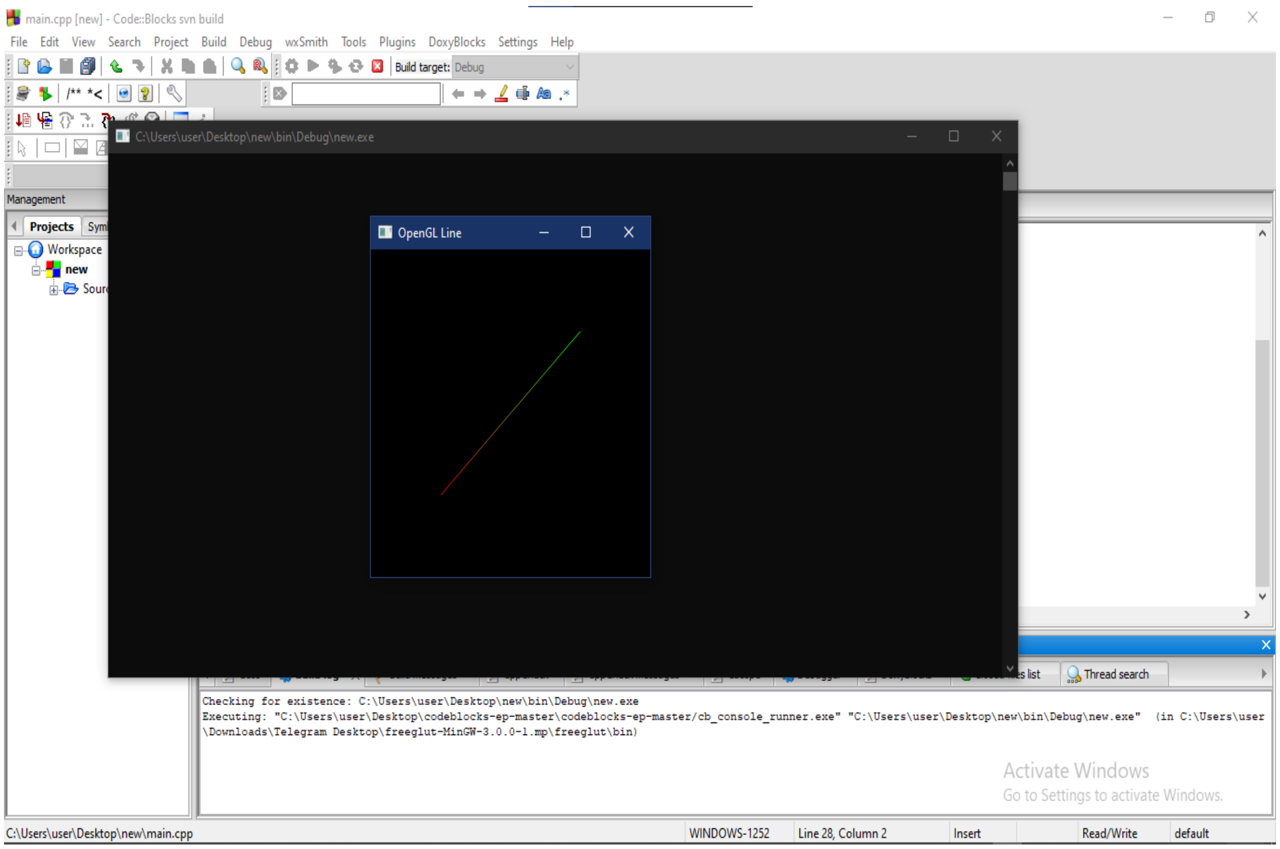
Section 1:OpenGL Initialization and Rendering

```
#include <GL/glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f); // Red color
    glVertex2f(0.0f, 0.5f);     // Top vertex
    glColor3f(0.0f, 1.0f, 0.0f); // Green color
    glVertex2f(-0.5f, -0.5f);    // Bottom-left vertex
    glColor3f(0.0f, 0.0f, 1.0f); // Blue color
    glVertex2f(0.5f, -0.5f);     // Bottom-right vertex
    glEnd();

    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("OpenGL Triangle");
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```


main.cpp [new] - Code::Blocks svn build
File Edit View Search Project Build Debug wxSmith Tools Plugins DoxyBlocks Settings Help
Build target: Debug
Management
Projects Symbols Files
Workspace
new
Sources
C:\Users\user\Desktop\new\bin\Debug\new.exe
OpenGL Triangle
Thread search
new.exe" (in C:\Users\user
Activate Windows
Go to Settings to activate Windows.
WINDOWS-1252 Line 31, Column 2 Insert Read/Write default

Opengl code for line

```
#include <GL/glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glColor3f(1.0f, 0.0f, 0.0f); // Red color
    glVertex2f(-0.5f, -0.5f);    // Start point
    glColor3f(0.0f, 1.0f, 0.0f); // Green color
    glVertex2f(0.5f, 0.5f);     // End point
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("OpenGL Line");
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutDisplayFunc(display);
    glutMainLoop();
        return 0;
}
```

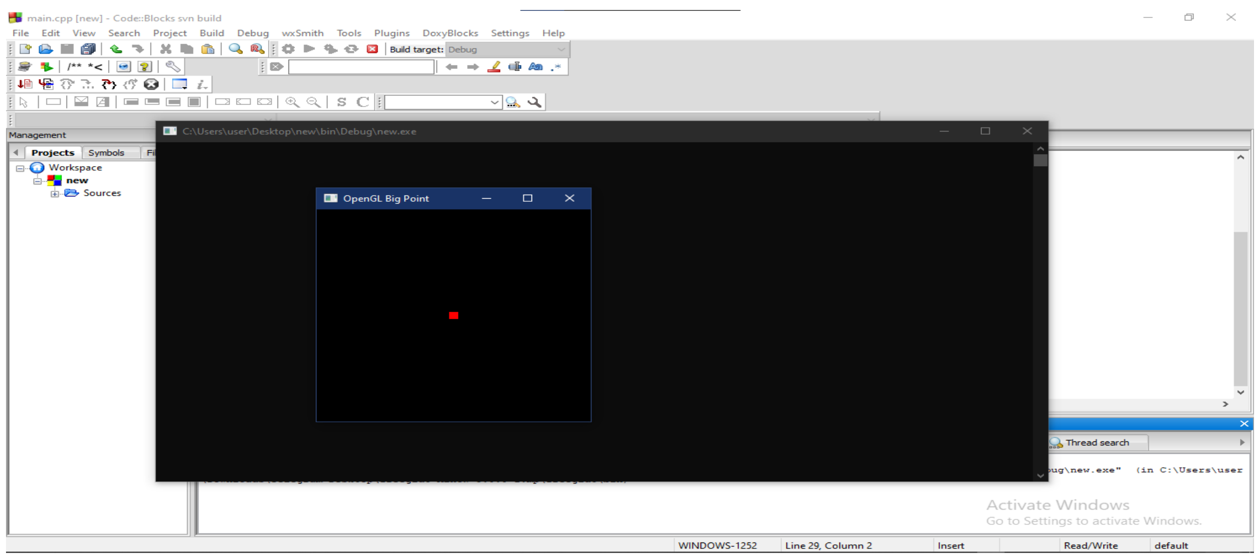

main.cpp [new] - Code::Blocks svn build
File Edit View Search Project Build Debug wxSmith Tools Plugins DoxyBlocks Settings Help
Build target: Debug
C:\Users\user\Desktop\new\bin\Debug\new.exe
OpenGL Line
Management
Projects
Workspace
new
Thread search
Activate Windows
Go to Settings to activate Windows.

```
Checking for existence: C:\Users\user\Desktop\new\bin\Debug\new.exe
Executing: "C:\Users\user\Desktop\codeblocks-ep-master\codeblocks-ep-master/cb_console_runner.exe" "C:\Users\user\Desktop\new\bin\Debug\new.exe"  (in C:\Users\user
\Downloads\Telegram Desktop\freeglut-MinGW-3.0.0-1.mp\freeglut\bin)
```

C:\Users\user\Desktop\new\main.cpp | WINDOWS-1252 | Line 28, Column 2 | Insert | Read/Write | default

Opengl code for point

```
#include <GL/glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(10.0f);  // Set the point size to a larger value
    glBegin(GL_POINTS);
    glColor3f(1.0f, 0.0f, 0.0f); // Red color
    glVertex2f(0.0f, 0.0f);  // Center point
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("OpenGL Big Point");
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```


main.cpp [new] - Code::Blocks svn build
File Edit View Search Project Build Debug wxSmith Tools Plugins DoxyBlocks Settings Help
Build target: Debug
Management
Projects Symbols
Workspace
new
Sources
C:\Users\user\Desktop\new\bin\Debug\new.exe
OpenGL Big Point
Thread search
Activate Windows
Go to Settings to activate Windows.
WINDOWS-1252 Line 29, Column 2 Insert Read/Write default

Here are the similarities and differences between rendering a point, triangle, and line with OpenGL and WebGL:

Similarities:

- **1. Rendering Principles:** Both OpenGL and WebGL follow similar principles for rendering graphics. They both utilize the concept of vertices and primitives (points, lines, and triangles) to define the geometry of objects.
- **2. Coordinate System:** OpenGL and WebGL use a similar Cartesian coordinate system, where the origin (0,0) is at the center of the screen or canvas, and the positive x-axis extends to the right while the positive y-axis extends upwards.
- **3. Color and Shading:** Both OpenGL and WebGL support setting colors for vertices and applying shading effects such as flat shading, Gouraud shading, or Phong shading to achieve realistic lighting effects.

Differences:

- **1. Platform:** OpenGL is a graphics rendering API primarily used for native applications on desktop and mobile platforms, while WebGL is a JavaScript API specifically designed for rendering graphics within web browsers. WebGL leverages the HTML5 canvas element and integrates seamlessly with web technologies.
- **2. Language and Integration:** OpenGL uses native programming languages (such as C/C++) and requires building and compiling separate applications. In contrast, WebGL uses JavaScript and can be directly embedded within web pages, allowing for easy integration with HTML, CSS, and other web technologies.
- **3. Context Creation:** In OpenGL, the context creation and management are handled by platform-specific libraries (such as GLUT or GLFW). In WebGL, the context is created through JavaScript APIs and is managed by the web browser.
- **4. Syntax and Function Calls:** The syntax and function calls differ between OpenGL and WebGL due to the differences in programming languages (e.g., C/C++ vs. JavaScript). While the core concepts remain the same, there are variations in the API calls and their parameters.

- **5. Security Restrictions:** WebGL operates within the security restrictions of web browsers, which impose limitations on accessing system resources, such as the GPU. OpenGL, being a native API, typically has more direct access to hardware and system resources.

These are some of the key similarities and differences between rendering a point, triangle, and line with OpenGL and WebGL. While the core concepts and principles are shared, the platform, language, integration, and syntax vary to accommodate the respective environments in which they are used.

Reference

- OpenGL Overview and OpenGL.org's Wiki with more information on OpenGL Language bindings
- OpenGL Architecture Review Board; Shreiner, Dave (2004). *OpenGL Reference Manual: The Official Reference Document to OpenGL*. Version 1.4. Addison-Wesley. ISBN 0-321-17383-X.
- OpenGL Architecture Review Board; Shreiner, Dave; et al. (2006). *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Version 2 (5th ed.). Addison-Wesley. ISBN 0-321-33573-2.
- "WebGL 2.0 Achieves Pervasive Support from all Major Web Browsers". *The Khronos Group*. 2022-02-09. Retrieved 2022-02-13.
- **^** "WebGL Specification". Khronos.org. Retrieved 2011-05-14.
- **^** "WebGL 2.0 Specification". Khronos.org. Retrieved 2017-02-27.
- "WebGL Fundamentals". HTML5 Rocks.
- **^** Parisi, Tony (2012-08-15). "WebGL: Up and Running". O'Reilly Media, Incorporated. Archived from the original on 2013-02-01. Retrieved 2012-07-13.

  ^ Jump up to:
- ***a b c*** "WebGL – OpenGL ES 2.0 for the Web". Khronos.org. Retrieved 2011-05-14.