



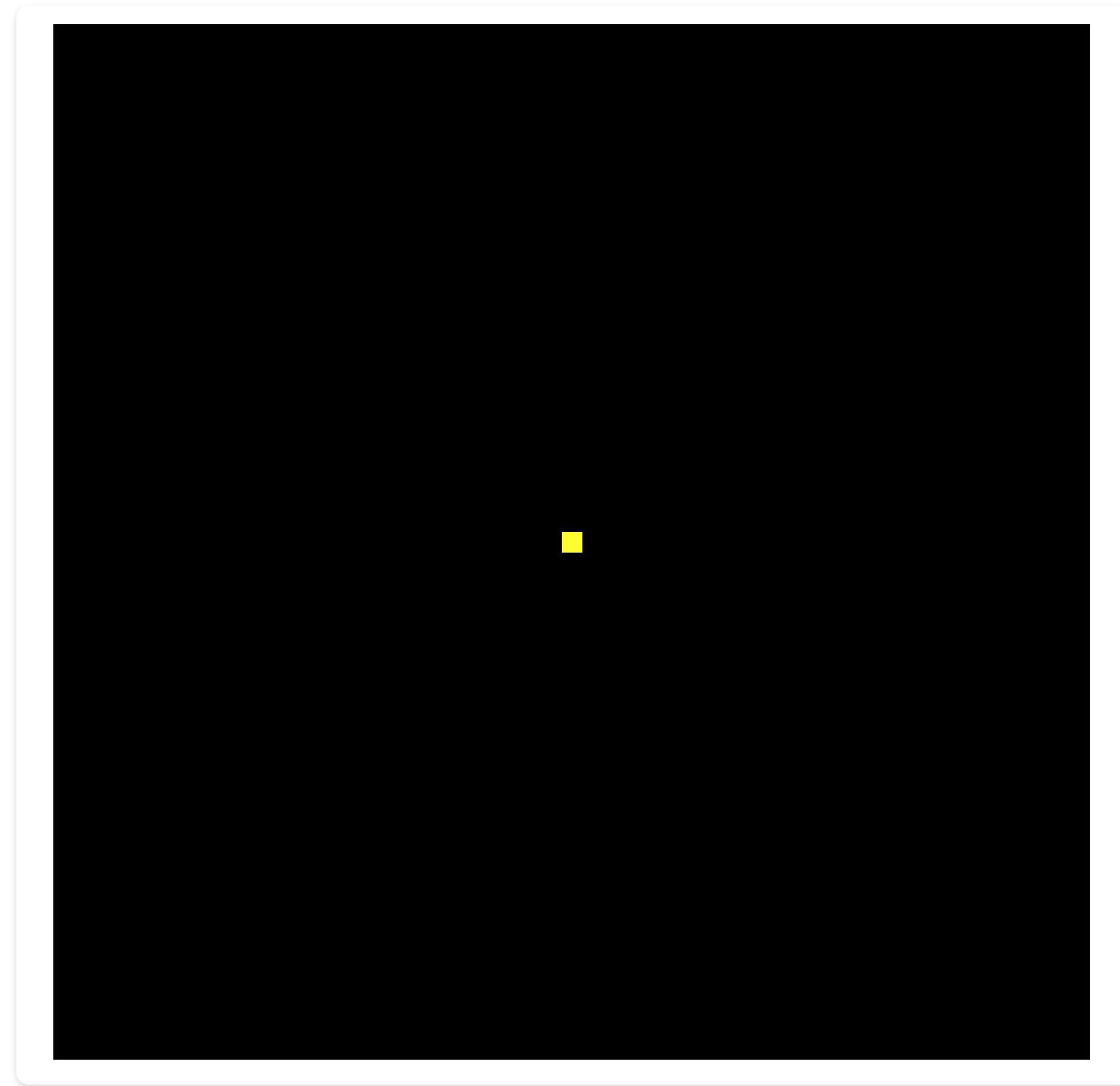
# WebGl

-johnnyzwu

- **Hello world -**      WebGL

- -
- -
- -
- -
- -

# Hello World



# index.ts

```
import { createProgramFromSource } from "../../utils";
import vertSource from "./index.vert";
import fragSource from "./index.frag";

function start() {
    const canvas = document.querySelector("canvas");
    canvas.width = 500;
    canvas.height = 500;

    // webgl
    const gl = canvas.getContext("webgl2");

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

    // webgl program
    const program = createProgramFromSource(gl, vertSource, fragSource);
    gl.useProgram(program);

    //
    gl.drawArrays(gl.POINTS, 0, 1);
}
```

# createShader

```
export function createShader(  
  gl: WebGL2RenderingContext,  
  type: number,  
  source: string  
) {  
  const shader = gl.createShader(type);  
  gl.shaderSource(shader, source);  
  gl.compileShader(shader);  
  
  const isOk = gl.getShaderParameter(shader, gl.COMPILE_STATUS)  
  
  if (isOk) return shader;  
  
  console.log(`  
    create ${type === gl.VERTEX_SHADER ? "vert" : "frag"} Sh  
    gl.getShaderInfoLog(shader)  
  `);  
  gl.deleteShader(shader);  
}  
}
```

# createProgram

```
export function createProgram(  
  gl: WebGL2RenderingContext,  
  vertexShader: WebGLShader,  
  fragmentShader: WebGLShader  
) {  
  const program = gl.createProgram();  
  gl.attachShader(program, vertexShader);  
  gl.attachShader(program, fragmentShader);  
  
  gl.linkProgram(program);  
  
  const isOk = gl.getProgramParameter(program, gl.LINK_STATUS)  
  
  if (isOk) return program;  
  
  console.log("createProgram failed:", gl.getProgramInfoLog(p  
  gl.deleteProgram(program);  
  gl.deleteShader(vertexShader);  
  gl.deleteShader(fragmentShader);  
}  
}
```

# createProgramFromSource

```
export function createProgramFromSource(  
  gl: WebGL2RenderingContext,  
  vertSource: string,  
  fragSource: string  
) {  
  const vertShader = createShader(gl, gl.VERTEX_SHADER, vertSource);  
  const fragShader = createShader(gl, gl.FRAGMENT_SHADER, fragSource);  
  
  return createProgram(gl, vertShader, fragShader);  
}
```

# index.vert

```
#version 300 es

void main() {

    gl_Position = vec4(0, 0, 0, 1);
    gl_PointSize = 10.0;
}
```

# index.frag

```
#version 300 es

precision highp float;

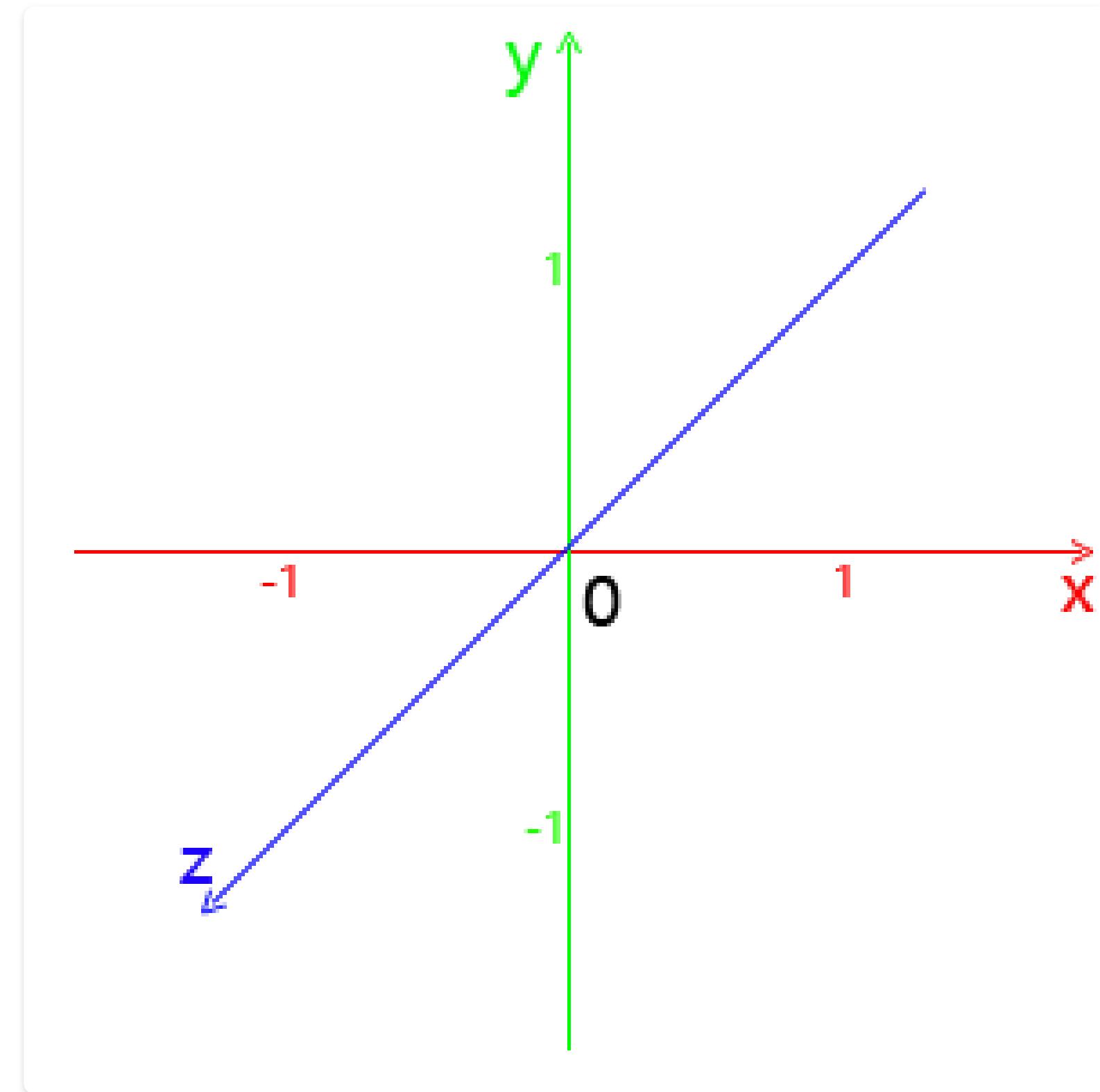
out vec4 outColor;

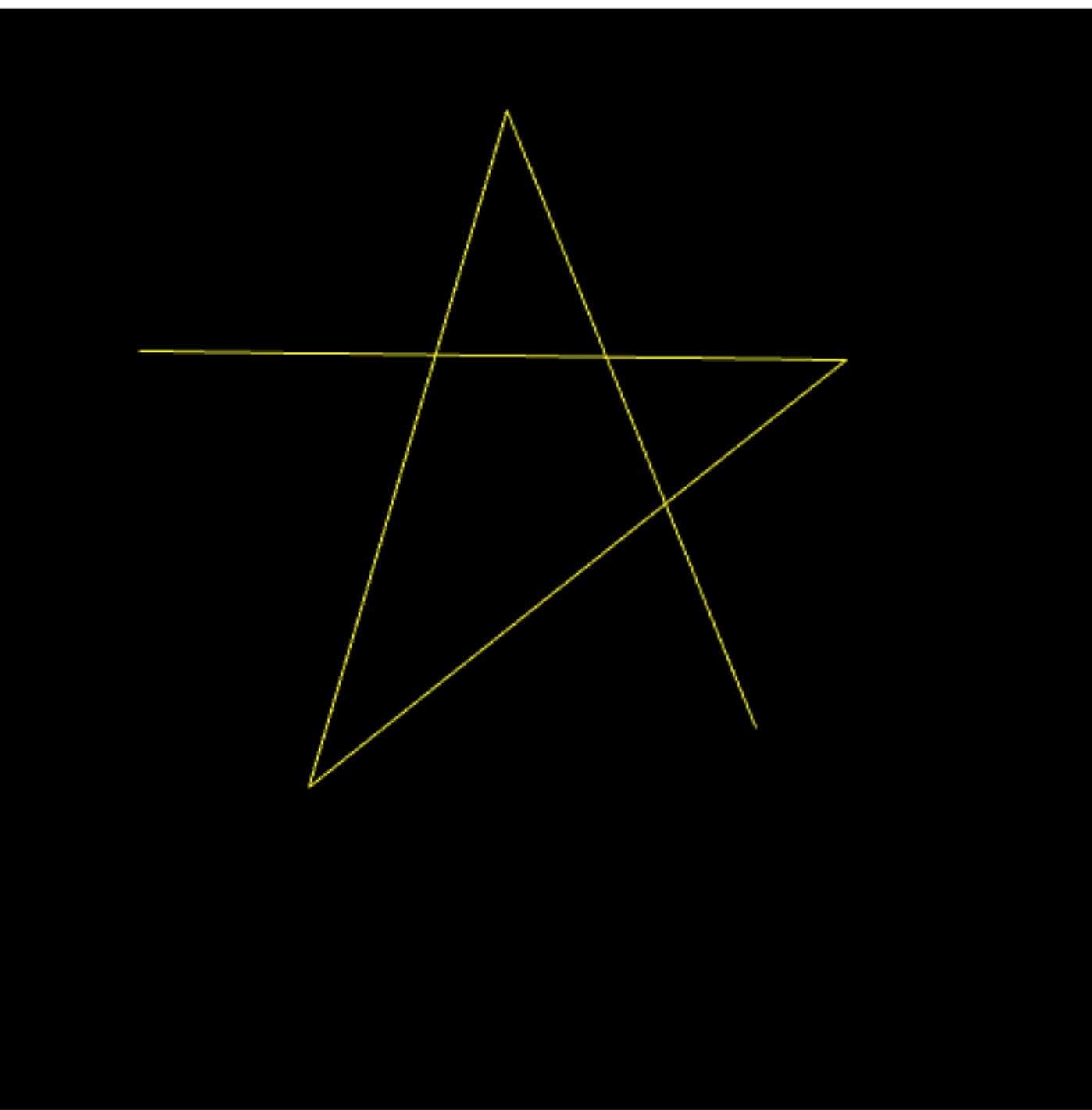
void main() {

    outColor = vec4(1, 1, 0, 1);
}
```

---

# WebGL





```
#version 300 es

in vec4 a_Position;

void main() {
    gl_Position = a_Position;
    gl_PointSize = 10.0;
}
```

# start

```
function start() {
  const canvas = document.querySelector("canvas");
  const gl = canvas.getContext("webgl2");

  const program = createProgramFromSource(gl, vertSource, fra )
  gl.useProgram(program);

  const buffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, buffer);

  const a_Position = gl.getAttributeLocation(program, "a_Positi
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0
  gl.enableVertexAttribArray(a_Position);

  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);

  const points = [];

  canvas.addEventListener("click", (e) => {
    handleClick(e, points, gl);
  });
}
```

# handleClick

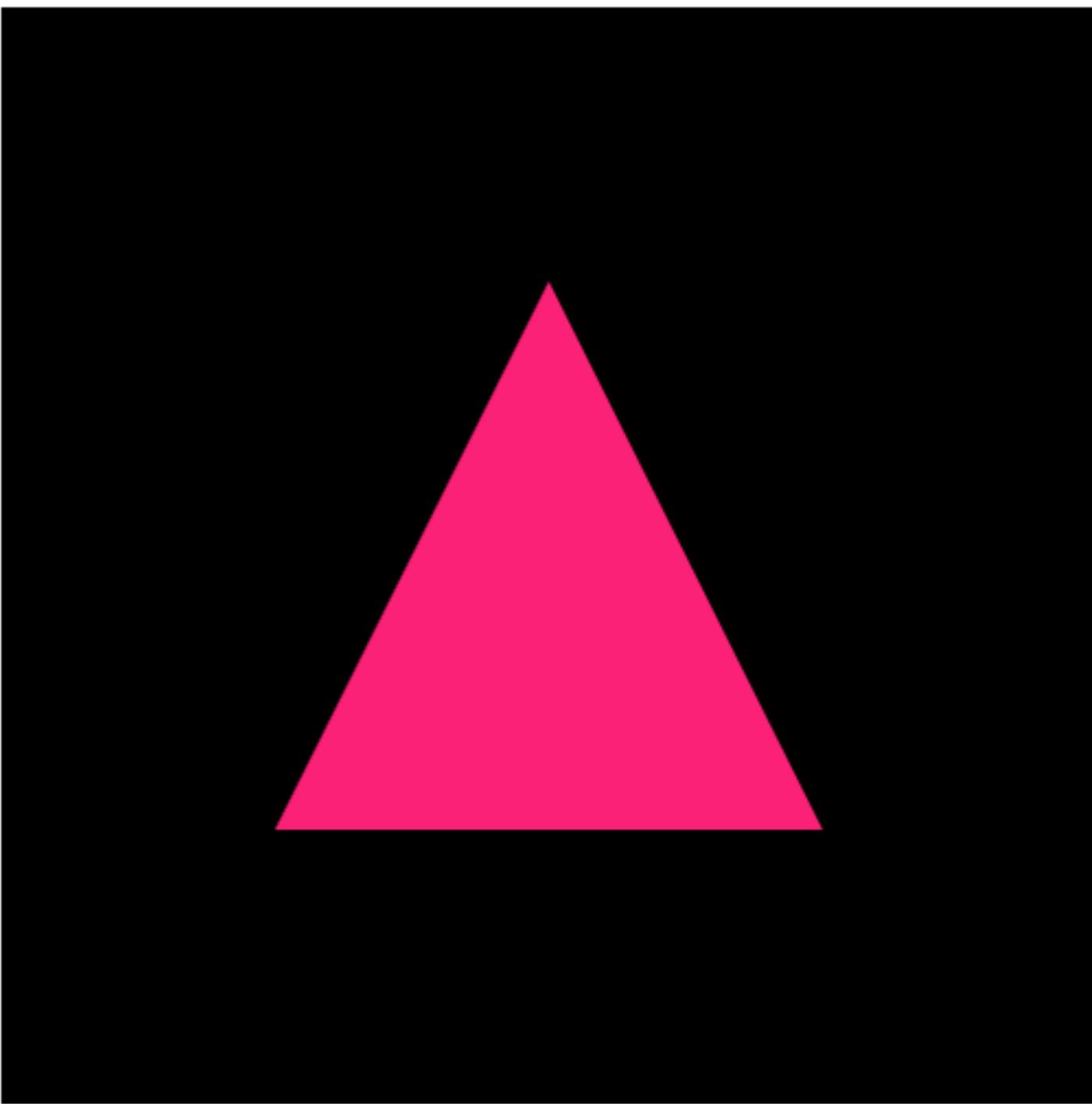
```
function handleClick(
  e: MouseEvent,
  points: number[],
  gl: WebGL2RenderingContext
) {
  const { clientX, clientY } = e;
  const canvas = e.target as HTMLCanvasElement;
  const { left, top } = canvas.getBoundingClientRect();

  const x = ((clientX - left) / canvas.width) * 2 - 1;
  const y = (((clientY - top) / canvas.height) * 2 - 1) * -1;
  points.push(x);
  points.push(y);
  const pointsLen = Math.floor(points.length / 2);

  const data = new Float32Array(points);
  gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);
  gl.clear(gl.COLOR_BUFFER_BIT);

  if (pointsLen < 2) {
    gl.drawArrays(gl.POINTS, 0, pointsLen);
  } else {
    gl.drawArrays(gl.LINE_STRIP, 0, pointsLen);
  }
}
```

```
1.      : `gl.createBuffer`  
2.      `gl.bindBuffer`  
3.      `gl.bufferData`  
4.      attribute `gl.vertexAttribPointer`  
5.      attribute `gl.enableVertexAttribArray`
```



# class Triangle

```
class Triangle {  
    readonly defaultPositionData = [  
        { x: -0.5, y: -0.5 },  
        { x: 0.5, y: -0.5 },  
        { x: 0, y: 0.5 },  
    ];  
  
    xTranslateCount = 0;  
    yTranslateCount = 0;  
    rotate = 0;  
    gl: WebGL2RenderingContext;  
  
    constructor(shaderSource: { vertSource: string; fragSource:  
        init() {}  
  
        handleTranslateX() {}  
  
        handleTranslateY() {}  
  
        handleRotate() {}  
  
        draw() {}  
    }  
}
```

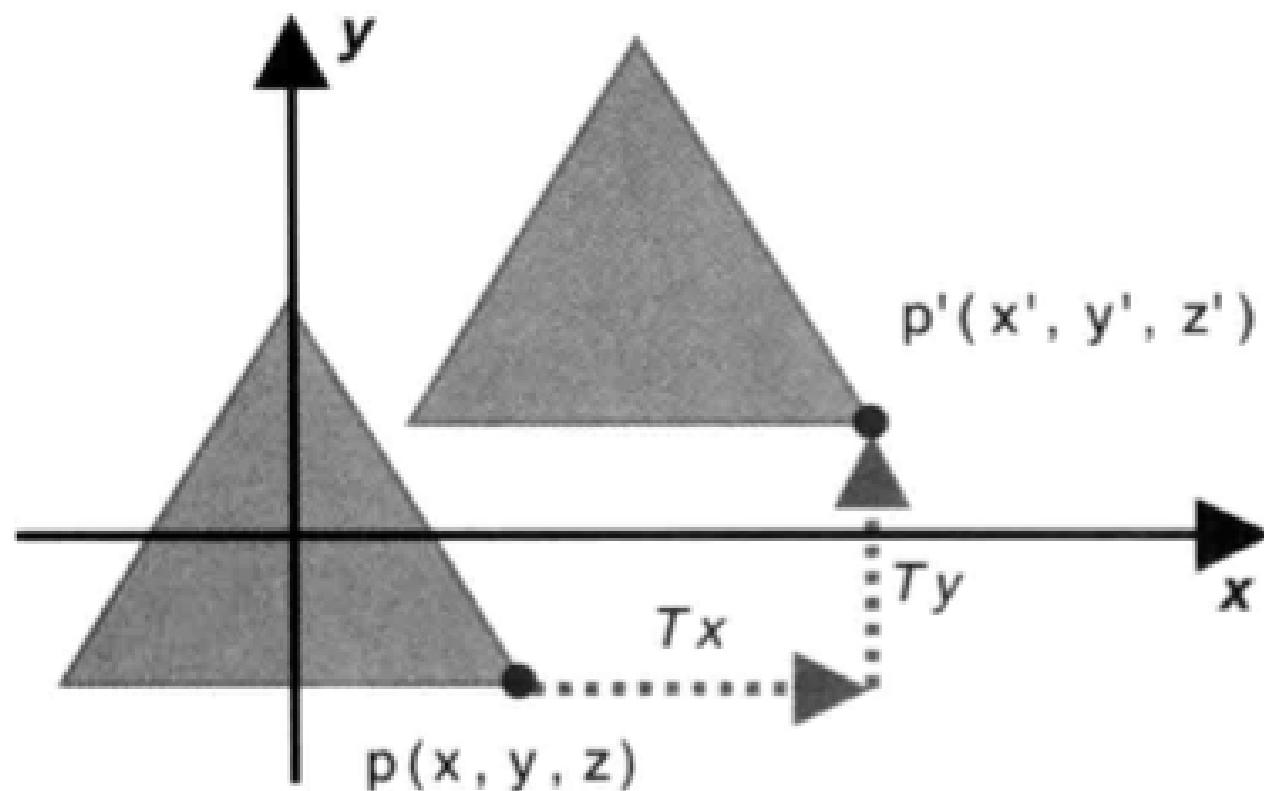
# method draw

```
draw() {  
    const finalData = this.defaultPositionData  
        .map(({ x, y }) => ({  
            x: x * Math.cos(this.rotate) - y * Math.sin(this.rotate),  
            y: x * Math.sin(this.rotate) + y * Math.cos(this.rotate)  
        }))  
        .map(({ x, y }) => ({  
            x: x + this.xTranslateCount,  
            y: y + this.yTranslateCount,  
        }))  
        .reduce((all, { x, y }) => [...all, { x, y }], []);  
  
    const gl = this.gl;  
  
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(finalData));  
  
    gl.clear(gl.COLOR_BUFFER_BIT);  
    gl.drawArrays(gl.TRIANGLES, 0, 3);  
}
```

■

■

■



- $x' = x + Tx$
- $y' = y + Ty$
- $z' = z + Tz$

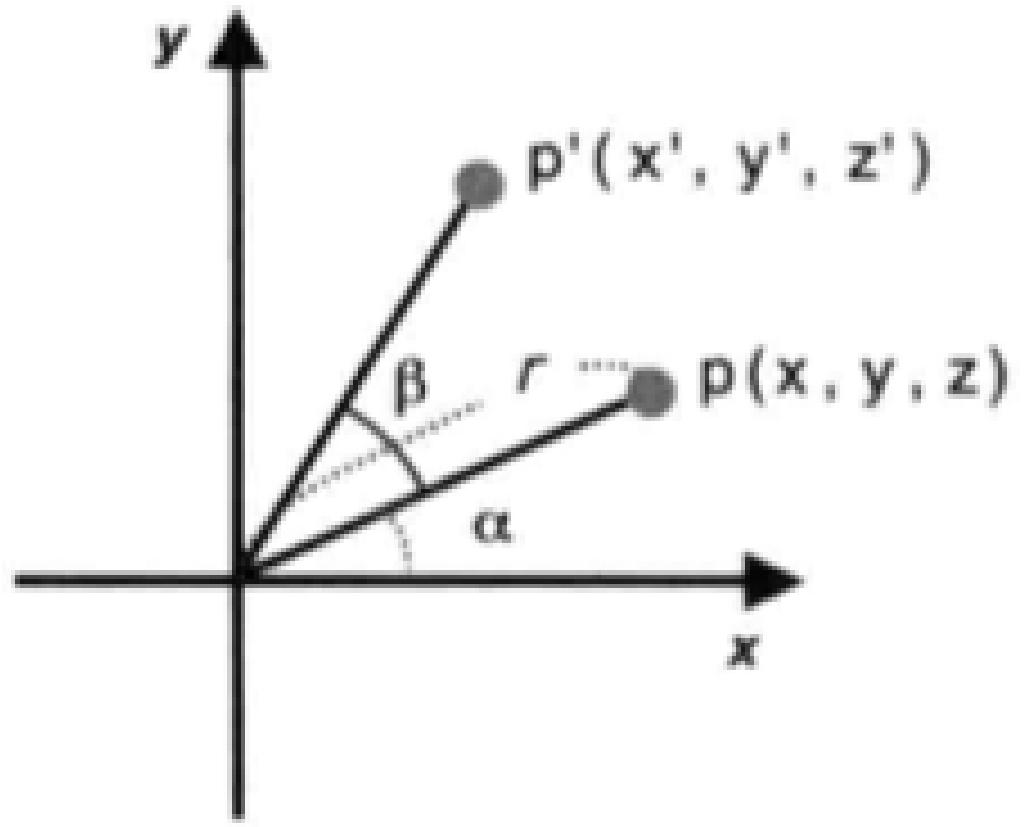
```

handleTranslateX() {
  const xInput = document.querySelector<HTMLInputElement>("input");
  xInput.addEventListener("input", (e: Event) => {
    const el = e.target as HTMLInputElement;
    const value = +el.value;

    this.xTranslateCount = value;

    this.draw();
  });
}

map(({ x, y }) => ({
  x: x + this.xTranslateCount,
  y: y + this.yTranslateCount,
}))
  
```



- $x = r \cos \alpha$
- $y = r \sin \alpha$
- $x' = r \cos (\alpha + \beta)$
- $y' = r \sin (\alpha + \beta)$
- $\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$
- $\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$
- $x' = x \cos \beta - y \sin \beta$
- $y' = x \sin \beta + y \cos \beta$

# handleRotate

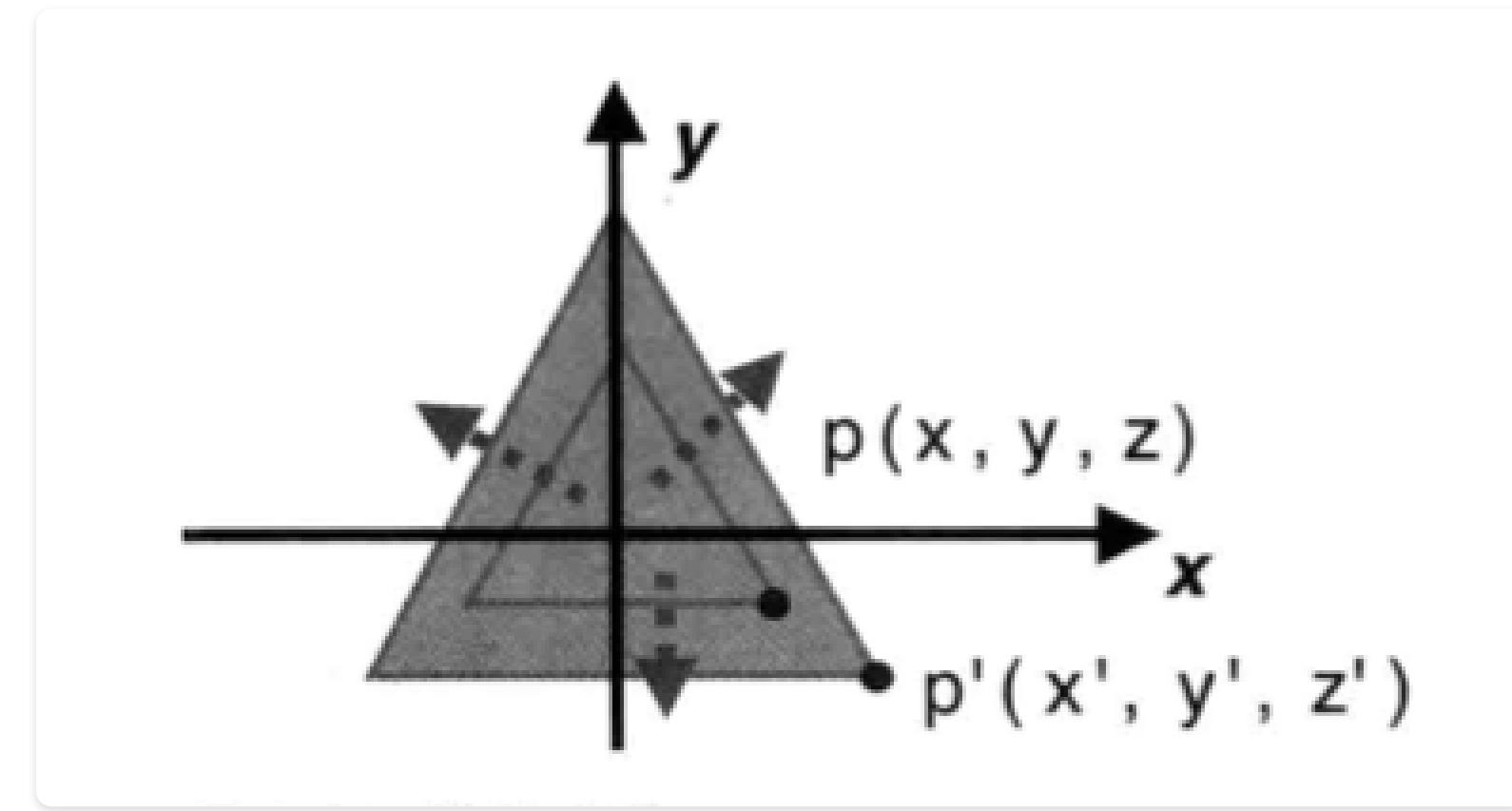
```
handleRotate() {
  const rInput = document.querySelector<HTMLInputElement>("#rotate");
  rInput.addEventListener("input", (e: Event) => {
    const el = e.target as HTMLInputElement;
    const value = +el.value;

    this.rotate = value * (Math.PI / 180);

    this.draw();
  });
}
```

# draw

```
map(({ x, y }) => ({
  x: x * Math.cos(this.rotate) - y * Math.sin(this.rotate),
  y: x * Math.sin(this.rotate) + y * Math.cos(this.rotate),
}))
```



- $x' = S_x \times x$
- $y' = S_y \times y$
- $z' = S_z \times z$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = 58$$

$$(1, 2, 3) \bullet (7, 9, 11) = 1 \times 7 + 2 \times 9 + 3 \times 11 = 58$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{aligned} a &= \cos\beta & b &= -\sin\beta & c &= 0 \\ d &= \sin\beta & e &= \cos\beta & f &= 0 \end{aligned}$$

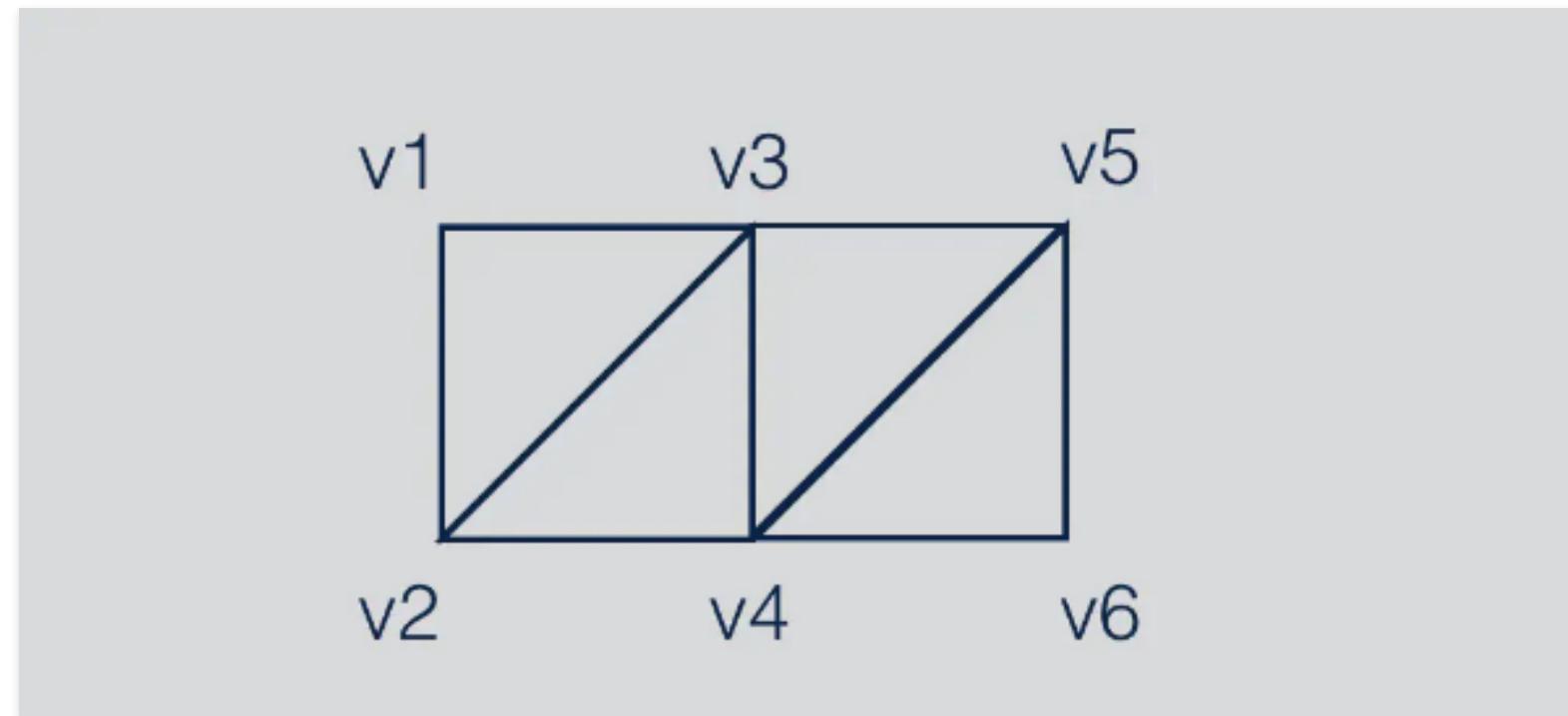
- $x' = ax + by + cz$   $g = 0$   $h = 0$   $i = 1$
- $y' = dx + ey + fz$
- $z' = gx + hy + iz$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- $x' = x \cos \beta - y \sin \beta$
- $y' = x \sin \beta + y \cos \beta$
- $z' = z$



## TRIANGLE\_STRIP

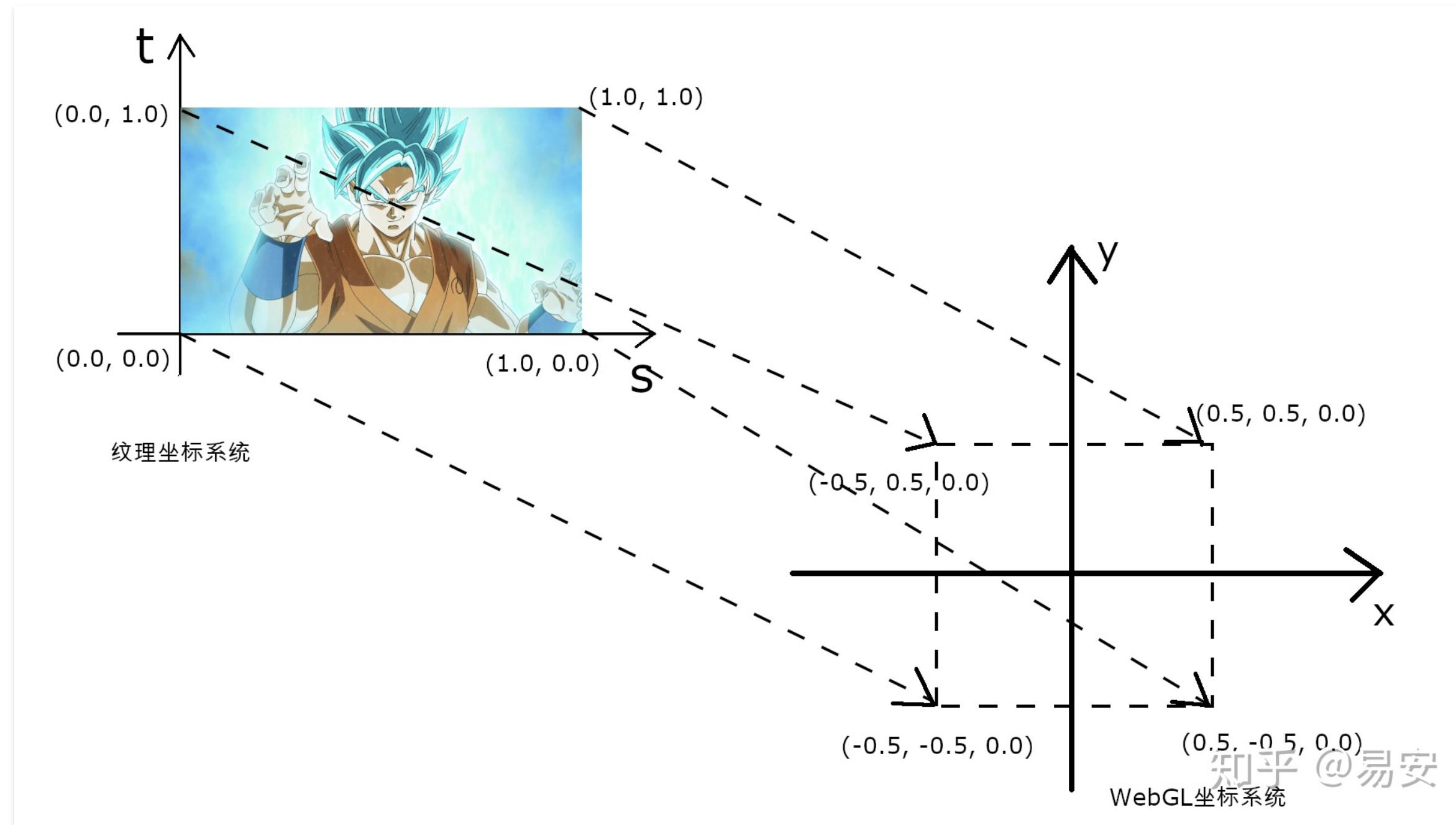


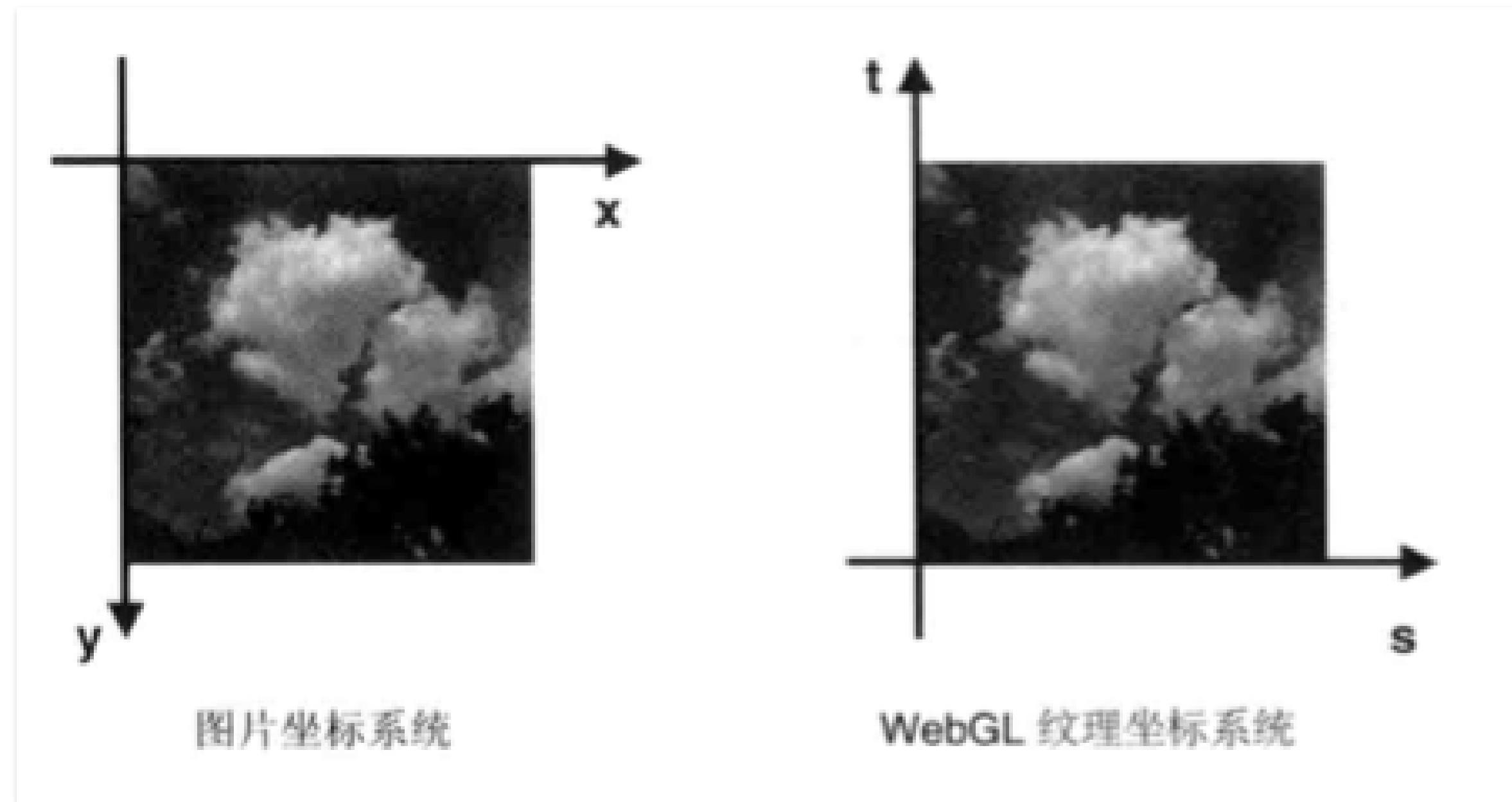
```
const data = new Float32Array([-1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, -1.0]);
gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);

// .....

gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
```

# webgl





```
#version 300 es

in vec4 a_position;
in vec2 a_texCoord;
out vec2 v_texCoord;

void main() {

    gl_Position = a_position;

    vTexCoord = a_texCoord;
}
```

```
#version 300 es

precision highp float;
uniform sampler2D u_image;

in vec2 vTexCoord;

out vec4 outColor;

void main() {
    outColor = texture(u_image, vTexCoord);
}
```

```
class ImageAdjust {
    initPosition(program: WebGLProgram) {
        const gl = this.gl;

        const data = new Float32Array([-1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, -1.0]);
        // ...
        gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);
    }

    initTexCoord(program: WebGLProgram) {
        const gl = this.gl;

        const data = new Float32Array([0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0]);
        //...
        gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);
    }
}
```

```
class ImageAdjust {
    initTexture(program: WebGLProgram, image: HTMLImageElement) {
        const gl = this.gl;

        const texture = gl.createTexture();
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);
        gl.activeTexture(gl.TEXTURE0);
        gl.bindTexture(gl.TEXTURE_2D, texture);

        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);

        const imageLocation = gl.getUniformLocation(program, "u_image");

        gl.uniform1i(imageLocation, 0);
    }
}
```

1. RGB, HSL, HSV

2.

- WebGL
- [webgl2fundamentals](#)
- [The Book of Shaders](#)
- [webgl water](#)