A 3D Cube in WebGL

We extend the previous example to create a rotating cube in 3D space.

Recap: What have we learned?

- We created a simple 3D WebGL application.
- We learned how to prepare the 3D data for the GPU.
- We learned how to apply 3D transformations to the data.
- We learned how to use shaders to render two 3D triangles.
- We learned to execute the depth obeying drawing commands.

Recap: What did we not do?

- We did not a real 3D object, but only two triangles.
- We did not animate the scene.

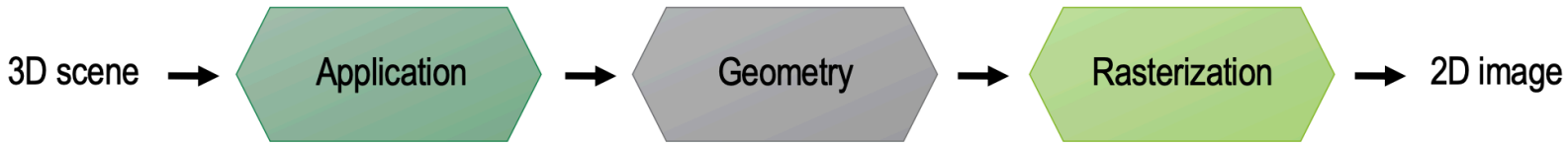
Explanation

👀 means that the code is already in the repository and you just need to look at it.
📋 means you can copy-paste the code and it should work.
📝 means that you need to create a new file
⭕ indicates that you need to do more than just copy-paste the code.
❌ indicates that you need to replace the old code with something new.

In any case you need to understand what you are doing.

The Rendering Pipeline

1. **Application** — your JavaScript code.
2. **Geometry** — defines shapes (points, lines, triangles).
    i. **Vertex Shader** — processes each vertex.
3. **Rasterization** — converts geometry to pixels.
    i. **Fragment Shader** — determines pixel color.
4. **2D image** — we need to display the result.



Application

WebGL Setup

👀 Start with an HTML canvas:

```
<canvas id="myCanvas" width="800" height="600"></canvas>
```

👀 Connect JavaScript using:

```
<script src="script.js" type="module"></script>
```

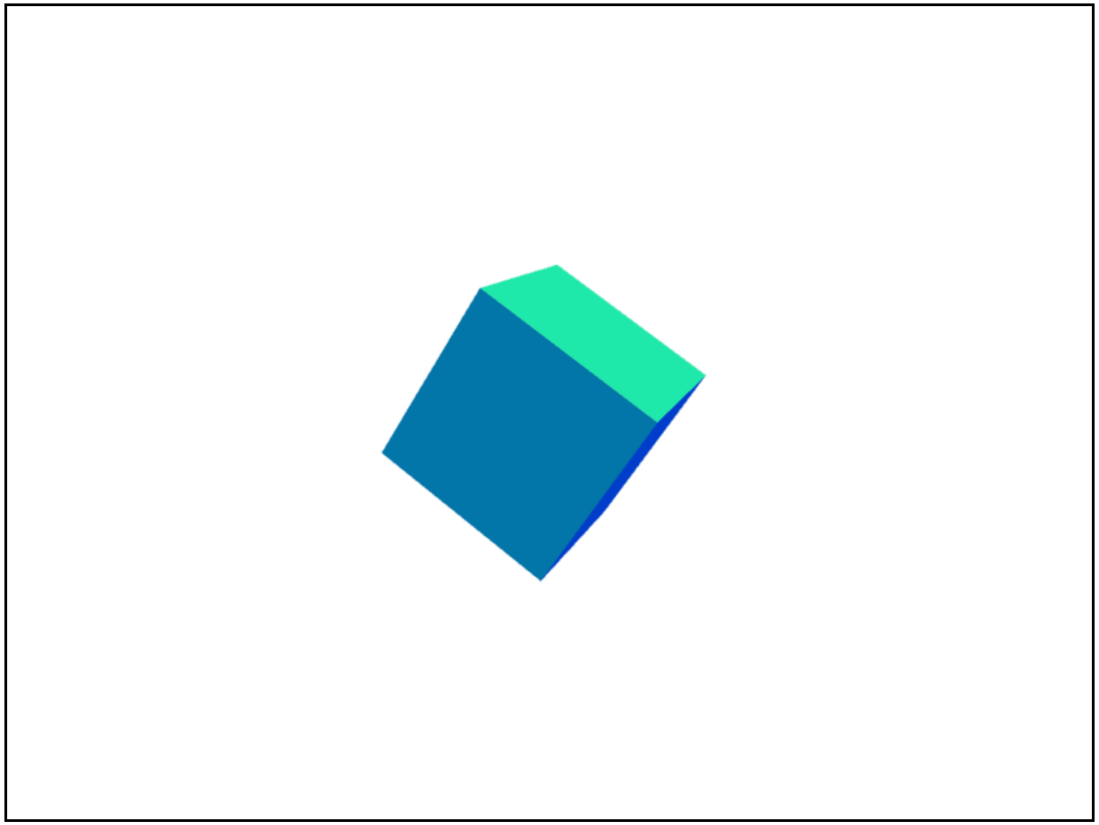
Initializing WebGL2

👀 Get the canvas element and create a **WebGL2** context.

```
const canvas = document.getElementById("myCanvas");
const gl = canvas.getContext("webgl2");

if (!gl) {
  console.error("WebGL2 not supported");
}
```

Geometry

The plan: a rotating cube



Define the cube geometry

We need to define many 3D coordinates, hence we define a class ❌📋

```
export default class Cube {
    constructor(size) {
        this.size = size;
        this.vertices = this.generateVertices();
        this.indices = this.generateIndices();
        this.colors = this.generateColors();
    }
    generateVertices() { ... }
    generateIndices() { ... }
    generateColors() { ... }
}
```

Import the cube class

We can import it in our main file with:

```
import Cube from "./cube.js";
```

Define cube vertices

⭕ The `generateVertices()` function shall create the vertices of a cube with the given size. The `TODO:` indicates that you need to add the other faces of the cube.

```
generateVertices() {
        const half = this.size / 2;
        return [
            // Front face (two triangles)
            -half, -half,  half,
             half, -half,  half,
             half,  half,  half,
            -half,  half,  half,
            // TODO: add the coordinates of the other faces
        ];
    }
```

Define cube indices

⭕ The `generateIndices()` function creates the indices for the cube. The `TODO:` indicates that you need to add the other faces of the cube.

```
generateIndices() {
        return [
            // Front face
            0, 1, 2,
            0, 2, 3,
            // TODO: add the indices of the other faces
        ];
    }
```

Define random cube colors

- 📋 The `generateColors()` function creates the colors for each vertex of the cube.
- Color each face with a different color, the cube has 6 faces, each face has 4 vertices, each vertex has 4 color components (RGBA)

```
generateColors() {
        const colors = [];
        for (let i = 0; i < 6; i++) {
            let color = [Math.random(), Math.random(), Math.random(), 1.0];
            colors.push(
                ...color,
                ...color,
                ...color,
                ...color
            );
        }
        return colors;
    }
```

Uploading position, indices and color to GPU

- ⭕ Keep the position and color buffer, but add another buffer for the indices

```
const indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cube.indices), gl.STATIC_DRAW);
```

Drawing to the Screen

❌ 📋 Now we need to tell WebGL to use the indices buffer to draw the triangles.

```
gl.drawElements(mode, count, type, offset);
```

⭕ Update the `count` to match the correct number of **indices**.

- The `mode` is still `gl.TRIANGLES`.
- The `type` is `gl.UNSIGNED_SHORT` because we are using 16-bit indices.
- The `offset` is `0` because we are starting from the beginning of the indices buffer.

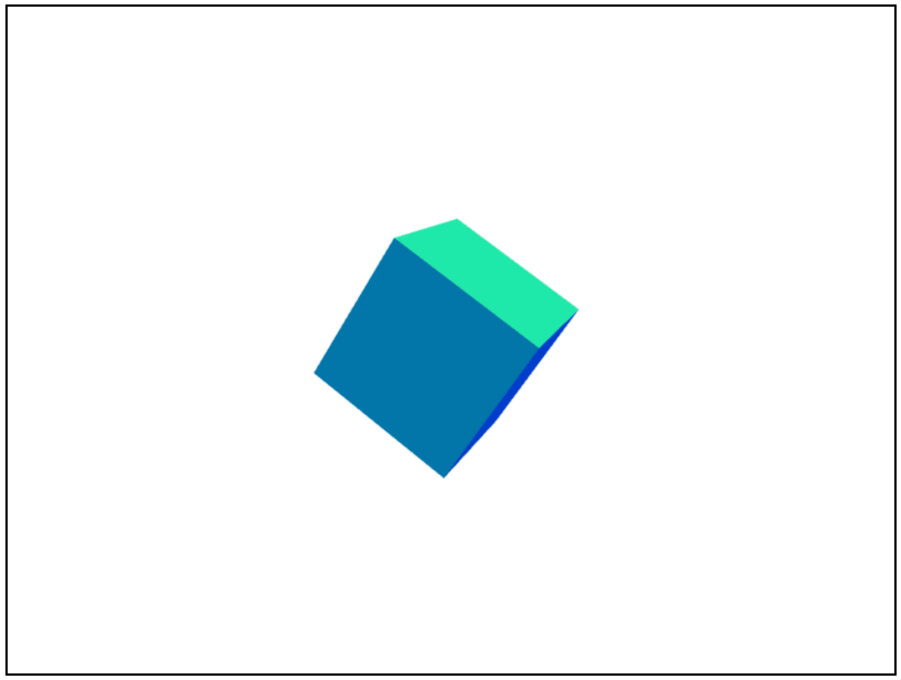
Animation

- ⭕ We need to add a rotation to the cube. We can do this by using a render loop.

```
// use a render loop to animate the cube
function render() {
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    // Update the model view matrix for animation
    mat4.rotate(modelViewMatrix, modelViewMatrix, 0.01, [1, 1, 0]); // Rotate slightly on each frame
    gl.uniformMatrix4fv(modelMatrixLocation, false, modelViewMatrix);

    // Draw the cube
    gl.drawElements(mode, count, type, offset);

    requestAnimationFrame(render);
}

// Start the render loop
render();
```

Result: Hello WebGL World

You should see the rotating cube on the canvas.



🎉 Congratulations, you've created your first animated WebGL 3D scene!