An interactive 3D Cube in WebGL

We extend the previous example and add controls to interact with the cube in 3D space.

Recap: What have we learned?

- We created a 3D WebGL application that renders an animated cube.
- We learned how to update 3D data and send it to the GPU.

Recap: What did we not do?

- We did not interact with the 3D object, we only animated it.

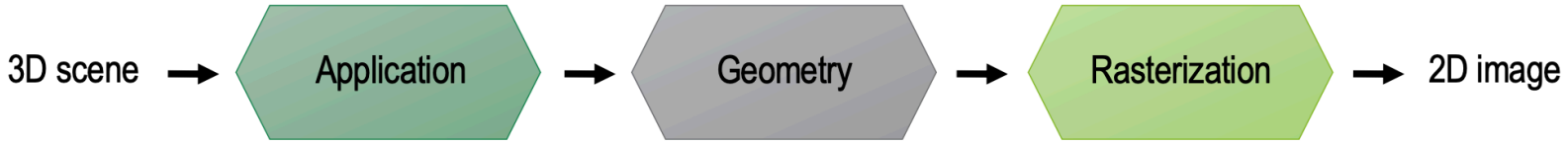
Explanation

👀 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>
```

HTML controls

📋 Add some HTML controls to interact with the cube:

```
<div class="sliders">
    <label>
        Rotate model about X axis:
        <input type="range" id="sliderX" min="-1" max="1" step="0.01"
            value="0" style="width: 150px;">
        <span id="valueX">0</span>
    </label>
    <label>
        Rotate model about Y axis:
        <input type="range" id="sliderY" min="-1" max="1" step="0.01"
            value="0" style="width: 150px;">
        <span id="valueY">0</span>
    </label>
</div>
```

- ⭕ You find some CSS in the `style.css` file to make it look nice. You find the CSS file in the `resources.zip` file that you can download from the course website.

Update the display

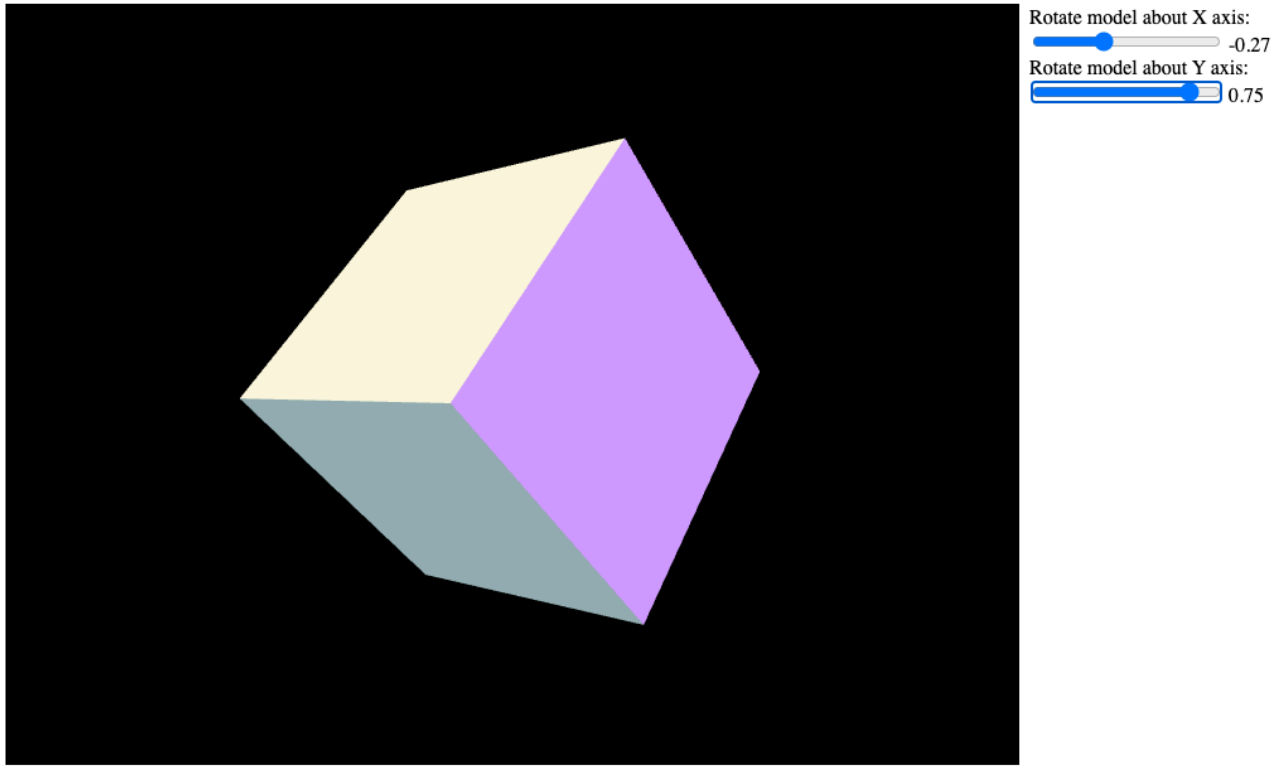
- We add a function to update the display when the sliders are moved.

```
document.getElementById("sliderX").addEventListener("input", function () {
    document.getElementById("valueX").textContent = this.value;
});
```

- ⭕ Add the same for the Y slider.

Geometry

The first plan: rotate the cube with the sliders



Updating the modelview matrix based on slider values

Add a new function that will be called, if the slider values change.

```
function updateMatrix() {
    let xValue = Number.parseFloat(document.getElementById("sliderX").value);

    let xRot = xValue * Math.PI;

    mat4.identity(modelViewMatrix);
    mat4.rotate(modelViewMatrix, modelViewMatrix, xRot, [1, 0, 0]);
    gl.uniformMatrix4fv(modelViewMatrixLocation, false, modelViewMatrix);
}
```

⭕ Call this function in your event handling methods.

⭕ Add the code to apply the rotation around the y-axis accordingly.

Projection matrix

👀 The projection matrix is the same as before.

```
// set the projection matrix
const fieldOfView = 45 * Math.PI / 180; // in radians
const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;
const zNear = 0.1;
const zFar = 100.0;
const projectionMatrix = mat4.create();
mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
```

Get the data to the GPU

👀 Take care that the modelview and projection matrices are still transferred to the shaders (and hence the GPU).

Drawing to the Screen

Prepare for drawing

❌ 📋 Prepare the drawing once:

```
// Clear the canvas and make the background black
gl.clearColor(0, 0, 0, 1);
gl.enable(gl.DEPTH_TEST);
gl.depthFunc(gl.LEQUAL);
```

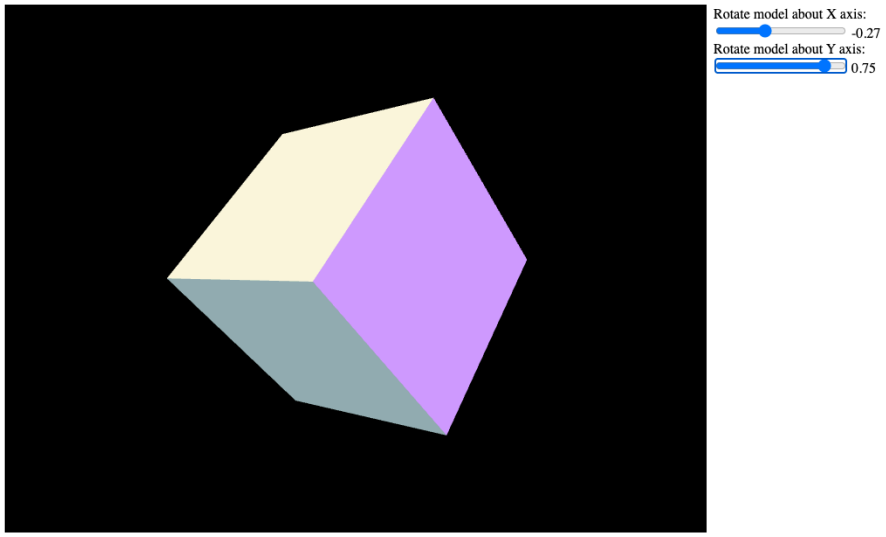
❌ 📋 We can simplify our render function now.

```
function render() {
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    // Draw the cube
    gl.drawElements(gl.TRIANGLES, cube.indices.length, gl.UNSIGNED_SHORT, 0);
    requestAnimationFrame(render);
}
```

Result: Interactive Cube

You should be able to rotate the cube on the canvas.



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

Interaction (cont.)

Mouse controls

⭕ Now add mouse controls to rotate the cube.

- You can use the `mousedown`, `mouseup` and `mousemove` events to get the mouse position and update the rotation accordingly.
- Note that you only need to update the sliders when the mouse is moved. They control the rotation of the cube.