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.

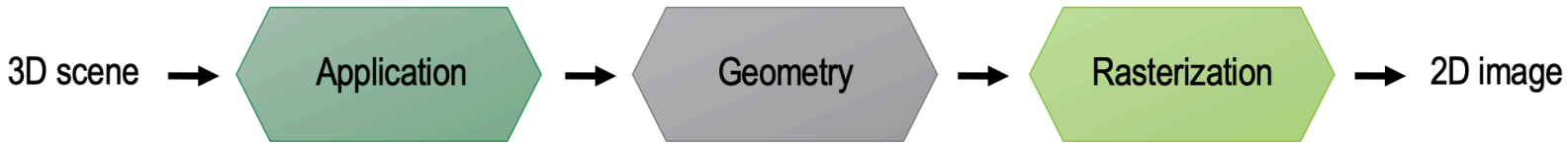
Explanation

👀 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

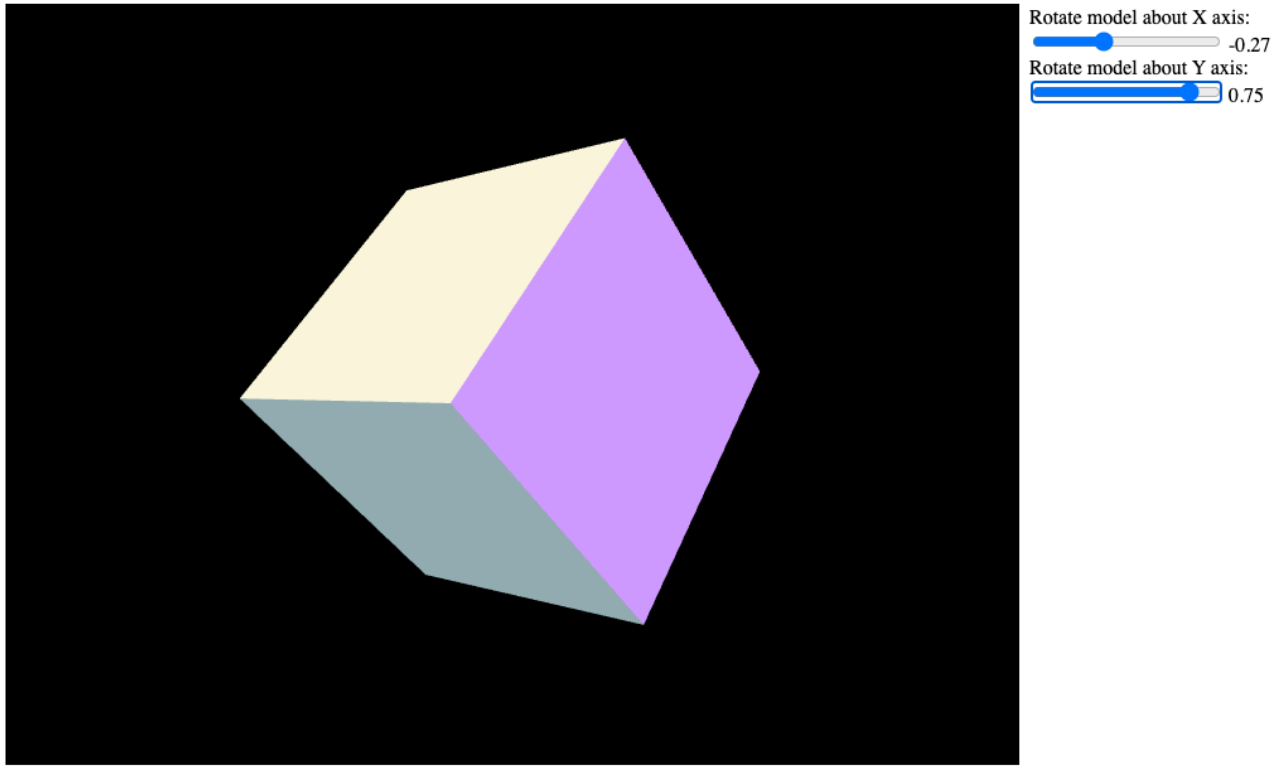
- We add a function to update the display when the sliders are moved.

```
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



Restructuring the code

📋 We need to restructure the code to make it easier to work with. We separate our transformation matrices into **model**, **view** and **projection** matrices.

- First we change the **vertex shader** to use these three matrices instead of just two.

```
uniform mat4 uModelMatrix;
uniform mat4 uViewMatrix;
uniform mat4 uProjectionMatrix;
```

- ⭕ Change the calculation of the `gl_Position` to use the new matrices.

Setting up the matrices initially (model)

❌ 📋 A new model matrix with small rotations around the x and y axes.

```
// set the model matrix
const modelMatrix = mat4.create();
mat4.rotateX(modelMatrix, modelMatrix, 0.3); // rotate around x-axis
mat4.rotateY(modelMatrix, modelMatrix, 0.2); // rotate around y-axis
```

Setting up the matrices initially (view)

❌ 📋 The view matrix is set up to look at the cube from a distance. We use the `lookAt` function to create the view matrix.

```
const viewPos = vec3.fromValues(0, 0, 3);
const viewTarget = vec3.fromValues(0, 0, 0);
const viewUp = vec3.fromValues(0, 1, 0);
const viewMatrix = mat4.create();
mat4.lookAt(viewMatrix, viewPos, viewTarget, viewUp);
```

Check the documentation and this or this explanation for more details.

Setting up the matrices initially (projection)

👀 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

❌ 📋 We need to send the matrices to the GPU. We do this by creating uniform variables in the shader and setting them in the JavaScript code.

```
gl.uniformMatrix4fv(modelMatrixLocation, false, modelMatrix);
gl.uniformMatrix4fv(viewMatrixLocation, false, viewMatrix);
gl.uniformMatrix4fv(projMatrixLocation, false, projectionMatrix);
```

Drawing to the Screen

Prepare for drawing

❌ 📋 We need to set up the viewport and clear the color and depth buffers.

```
// prepare everything for drawing
gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);
gl.clearColor(.0, .0, .0, 1); // dark background
gl.enable(gl.DEPTH_TEST);
gl.depthFunc(gl.LEQUAL);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.cullFace(gl.BACK);
gl.enable(gl.CULL_FACE);
```

Draw the cube

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

```
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.

Interaction

- ❌⭕ We need to remove the animation code from the previous example. But, we will still use a render loop as we want to update the values interactively.

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

    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    // Update the model matrix for animation
    let xRot = xValue * Math.PI;
    mat4.identity(modelMatrix);
    mat4.rotate(modelMatrix, modelMatrix, xRot, [1, 0, 0]);
    gl.uniformMatrix4fv(modelMatrixLocation, false, modelMatrix);

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

    requestAnimationFrame(render);
}
```

Interaction (cont.)

- ⭕ Add the same for the Y slider.
- Update the `modelMatrix` to use both sliders.
- Think about how to combine the two rotations.

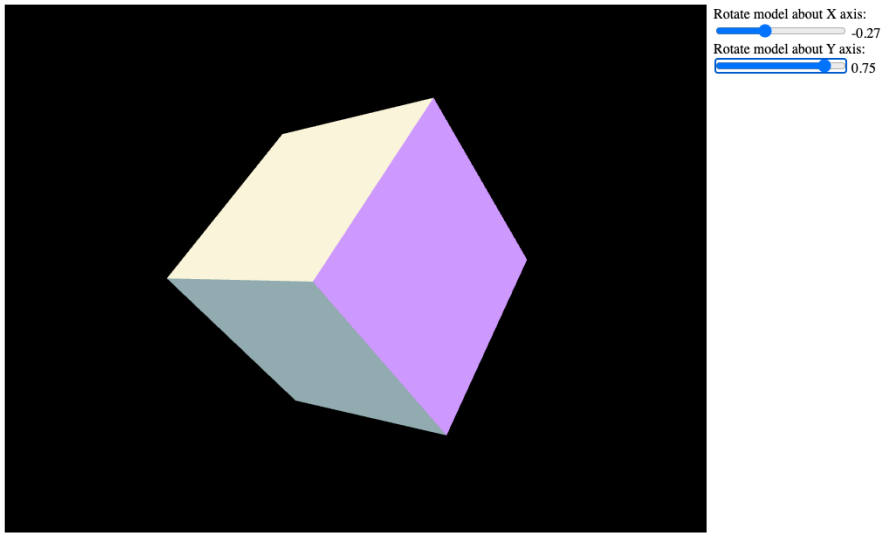
Mouse controls

⭕ 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.

Result: Interactive Cube

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



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