
OpenGL WebGL
HANDS ON

Interação com Objetos em Movimento

Aplicações OpenGL / WebGL

Application Code

OpenGL Commands (API_calls)

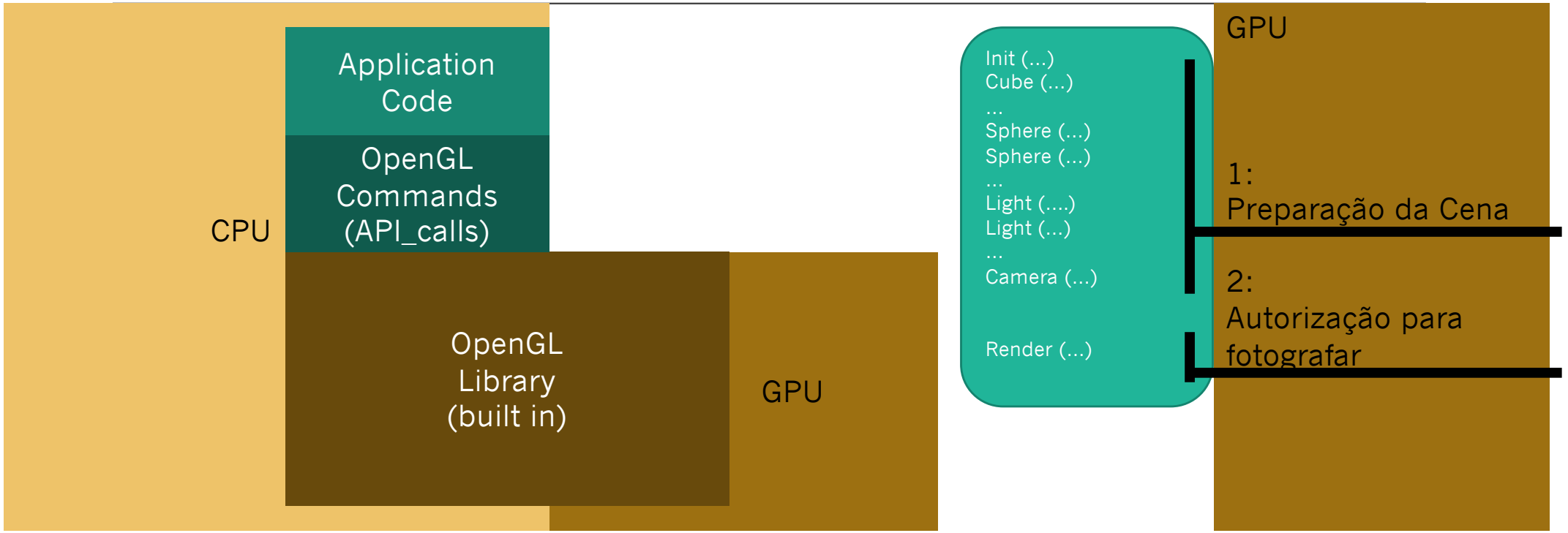
OpenGL: Máquina de Estados

Estado 1 – sendo configurada

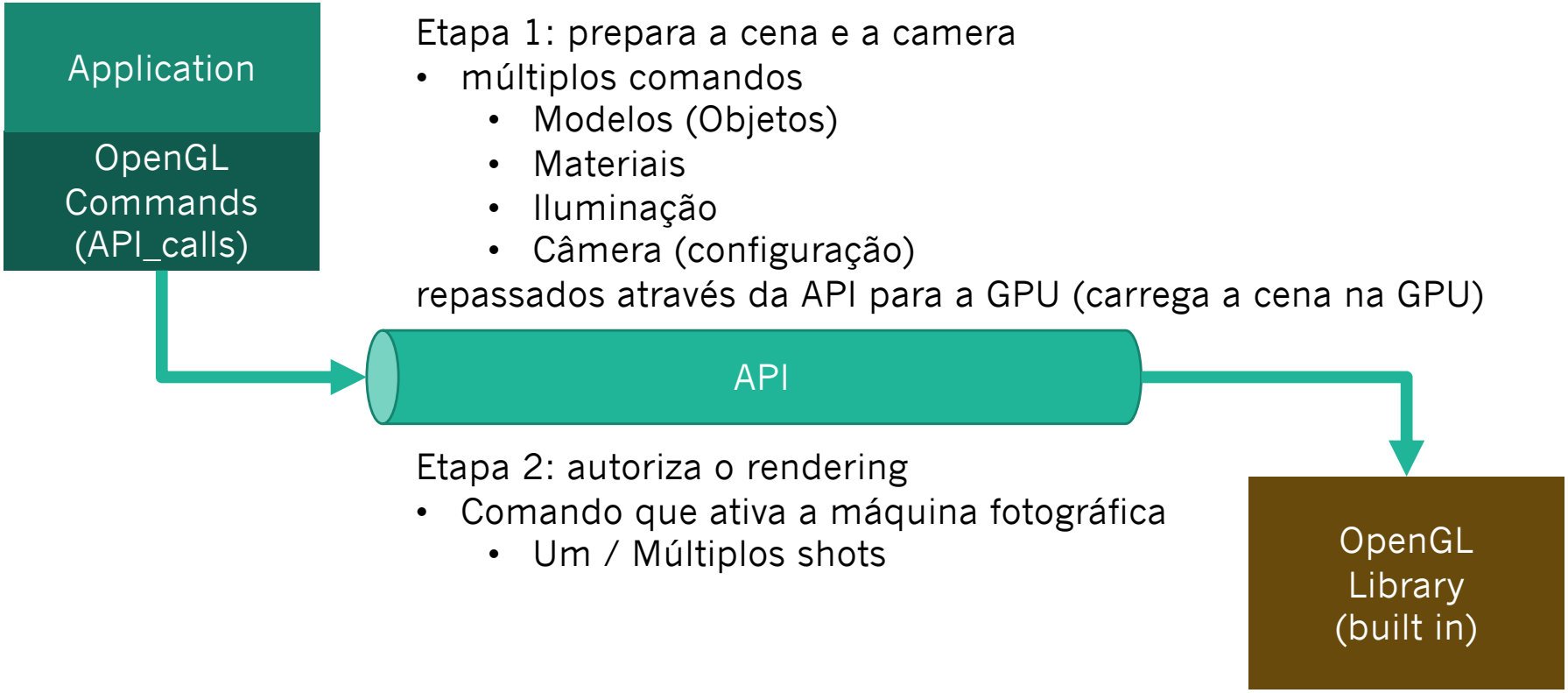
Estado 2 – gerando imagens

OpenGL Library (built in)

Aplicações OpenGL / WebGL



Aplicações
OpenGL / WebGL

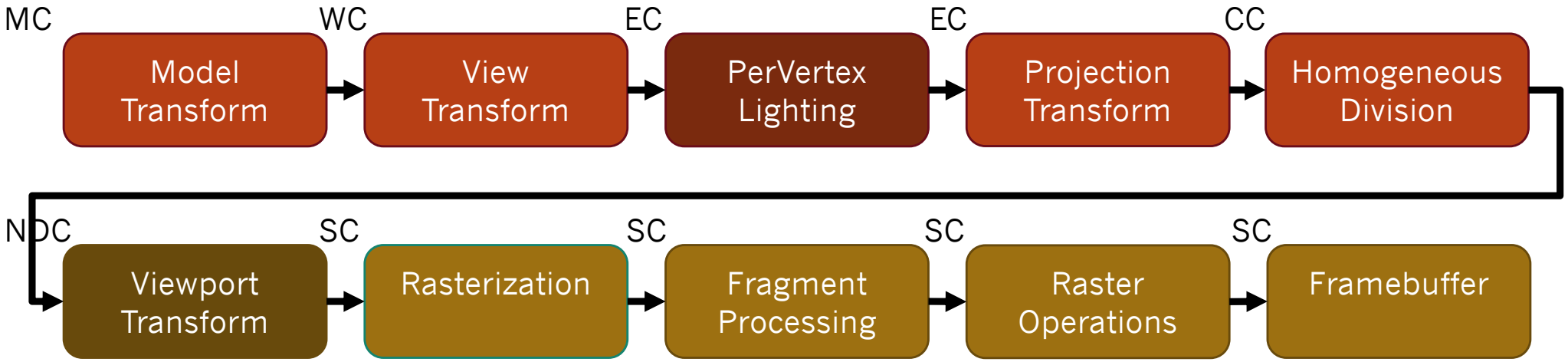


Etapa 1: prepara a cena e a camera
- múltiplos comandos
  - Modelos (Objetos)
  - Materiais
  - Iluminação
  - Câmera (configuração)

repassados através da API para a GPU (carrega a cena na GPU)

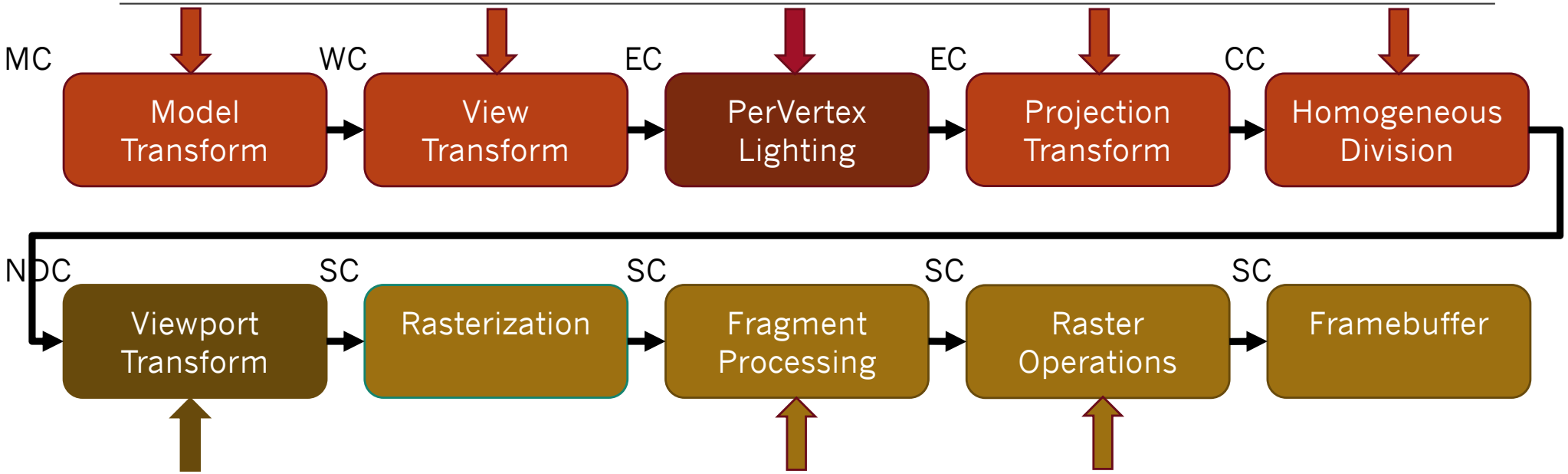
Etapa 2: autoriza o rendering
- Comando que ativa a máquina fotográfica
  - Um / Múltiplos shots

The graphics pipeline (object to picture) - fotografando um objeto virtual



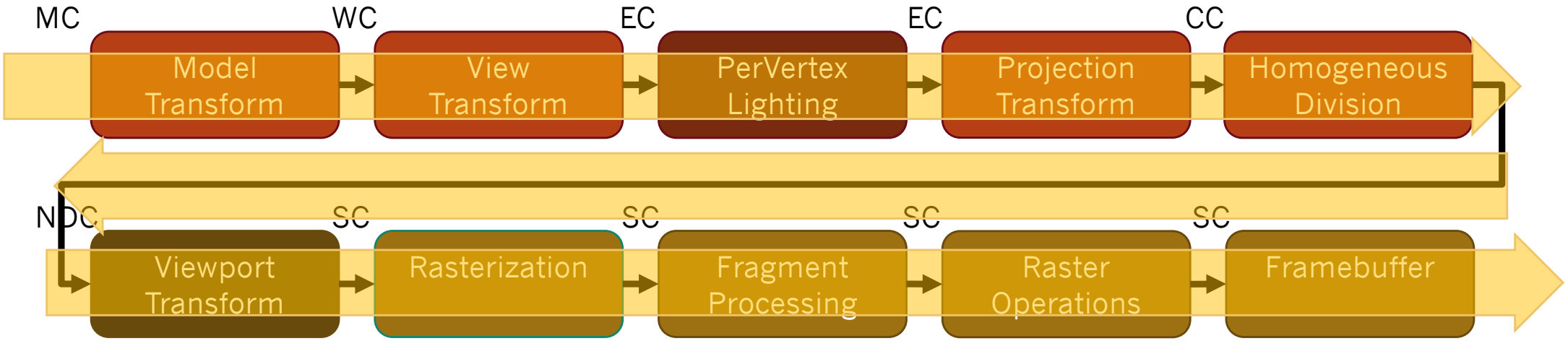
COORDINATES	MC	Model	CC	Clip
	WC	World	NDC	Normalized Device
	EC	Eye	SC	Screen

The graphics pipeline (object to picture) - fotografando um objeto virtual



COORDINATES				
	MC	Model	CC	Clip
	WC	World	NDC	Normalized Device
	EC	Eye	SC	Screen

The graphics pipeline (object to picture) - fotografando um objeto virtual



COORDINATES	MC	Model	CC	Clip
	WC	World	NDC	Normalized Device
	EC	Eye	SC	Screen

Listas

V [xyz / xyz / xyz ....]

- V1 / v2 / v3 implicito

[rgb / rgb / rgb .....]

F [vvv / vvv / vvv ...]

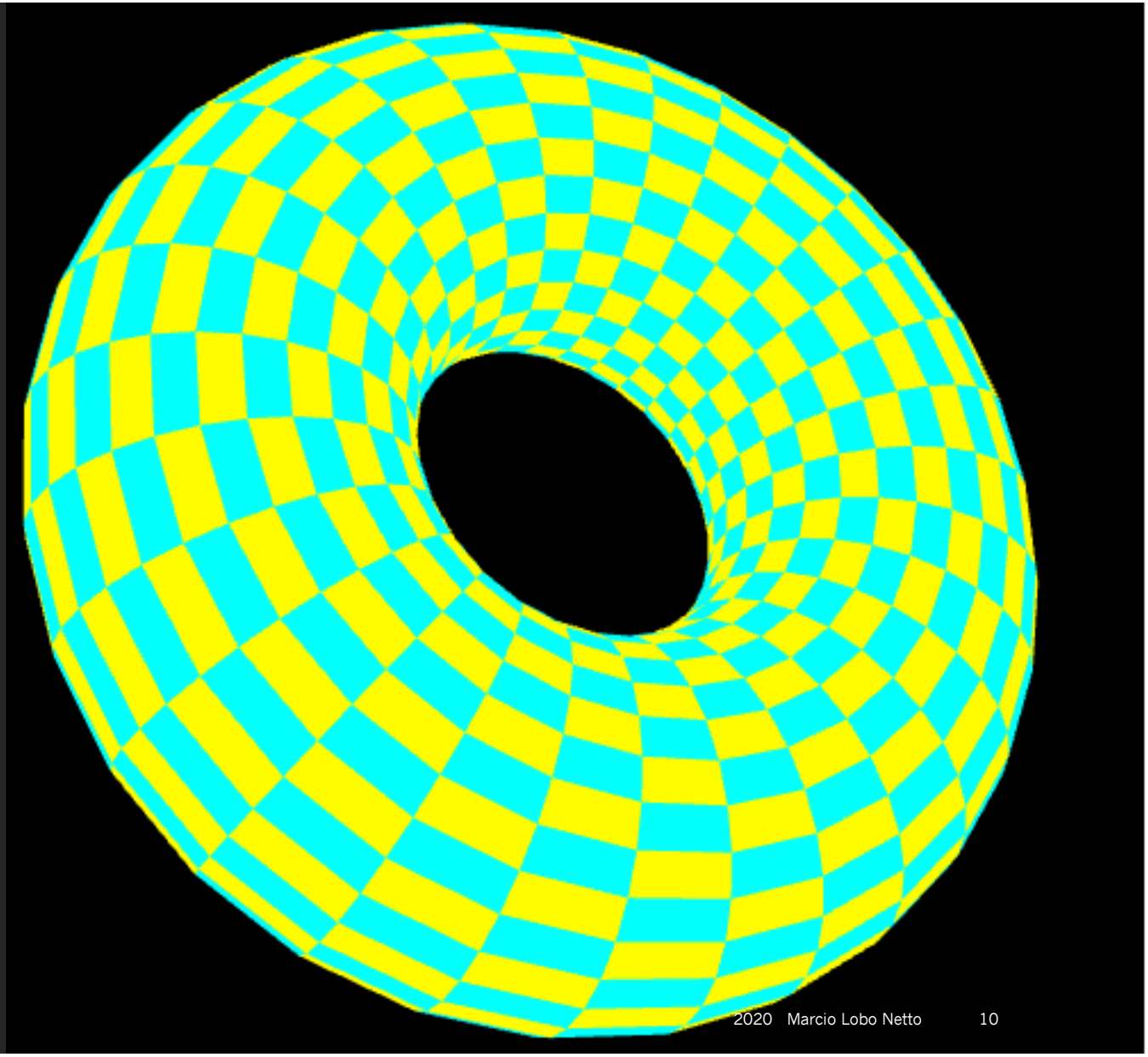
- f1 / f2 / f3 implicito

O [fffff / fffffffff / ffffff /ffffff ....]

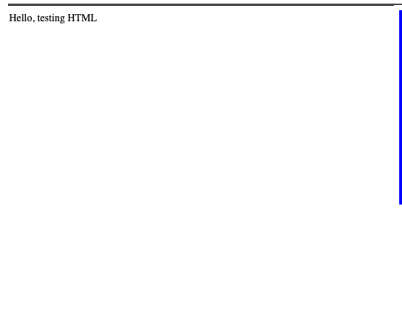
- o1 / o2 / o3 implicito

Material
Prof Hae

http://www.lps.usp.br/hae/apostilaCG/apostila-webgl/index.html

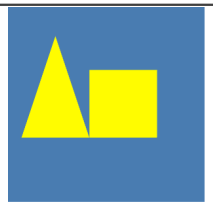


Cases



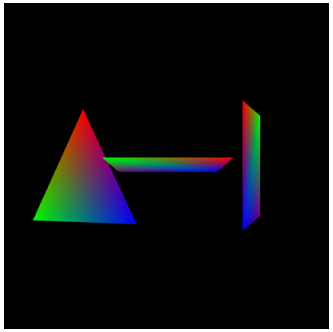
Texto

Canvas

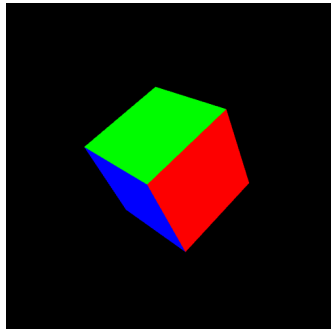


Objs 2D

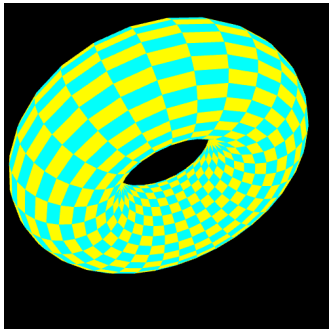
Objs 2D coloridos



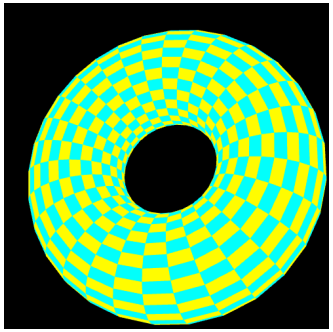
Objs 2D coloridos rodando 3D



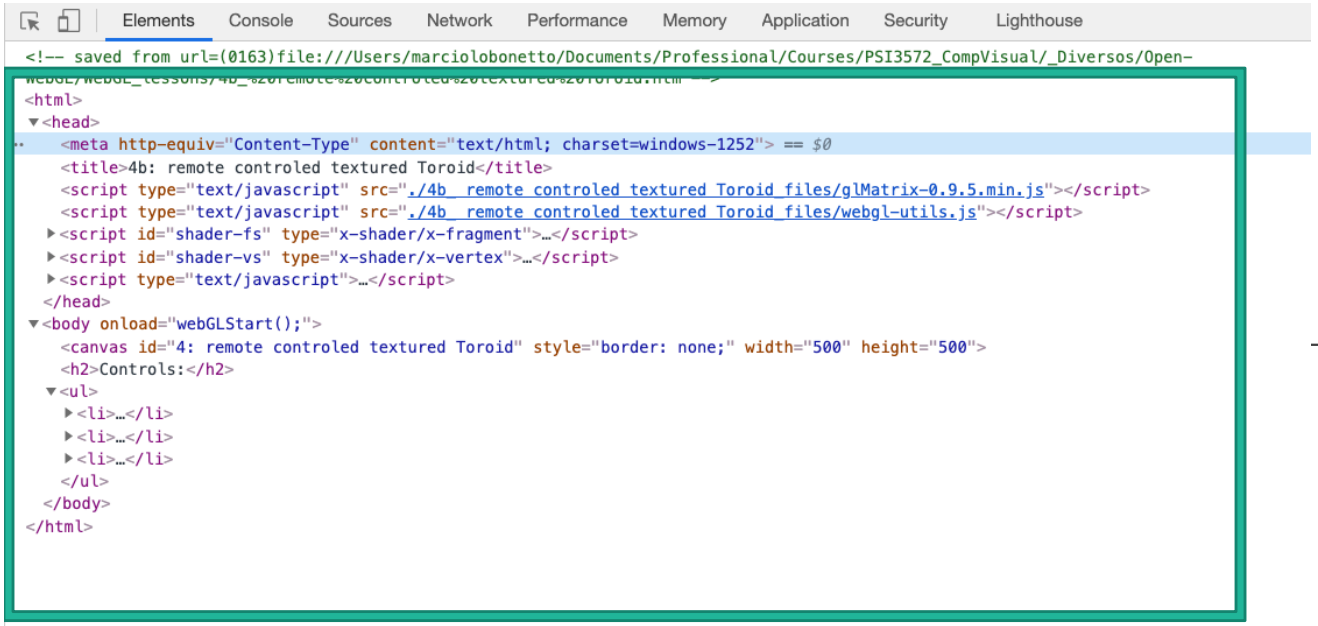
Obj 3D colorido rodando 3D



Obj 3D textura rodando 3D

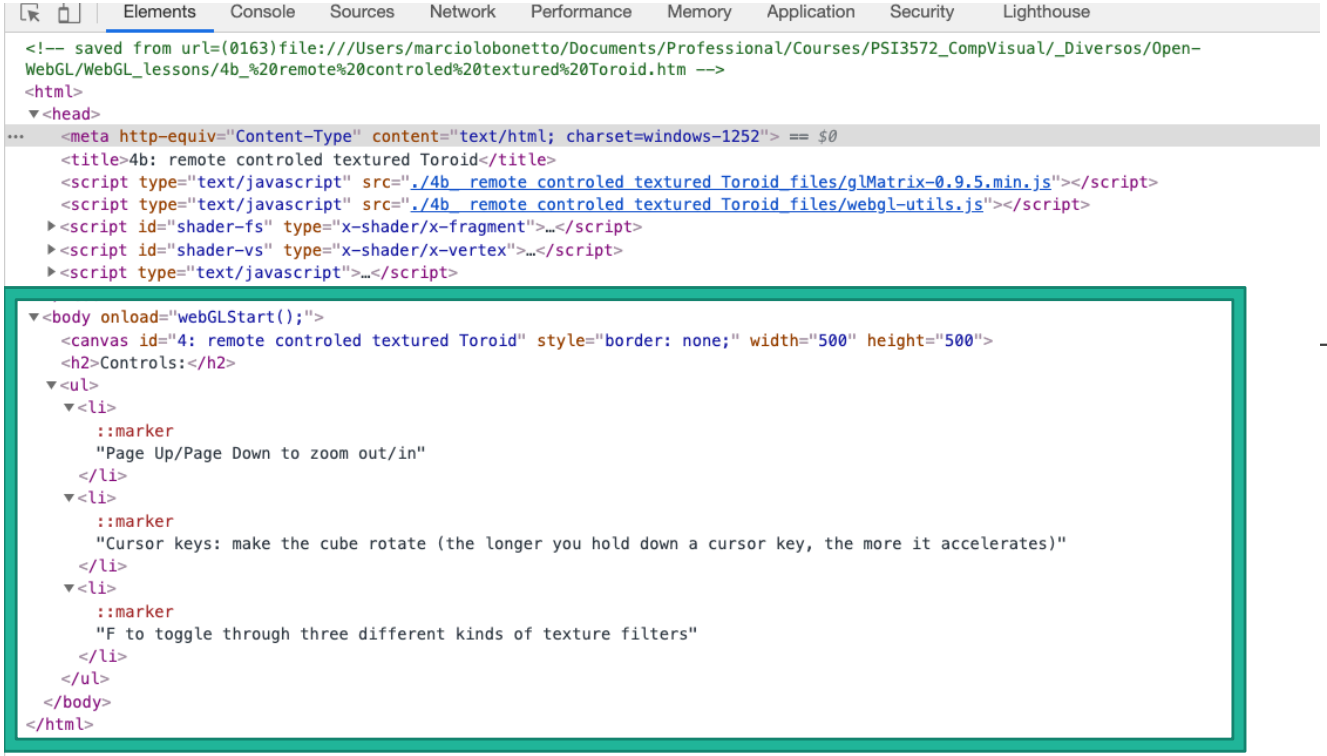


Obj 3D textura movimento controlado


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
<!-- saved from url=(0163)file:///Users/marciolobonetto/Documents/Professional/Courses/PSI3572_CompVisual/_Diversos/Open-
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252"> == $0
<title>4b: remote controled textured Toroid</title>
<script type="text/javascript" src="./4b_ remote controled textured Toroid_files/glMatrix-0.9.5.min.js"></script>
<script type="text/javascript" src="./4b_ remote controled textured Toroid_files/webgl-utils.js"></script>
<script id="shader-fs" type="x-shader/x-fragment">…</script>
<script id="shader-vs" type="x-shader/x-vertex">…</script>
<script type="text/javascript">…</script>
</head>
<body onload="webGLStart();">
<canvas id="4: remote controled textured Toroid" style="border: none;" width="500" height="500">
<h2>Controls:</h2>

…
…
…

</body>
</html>

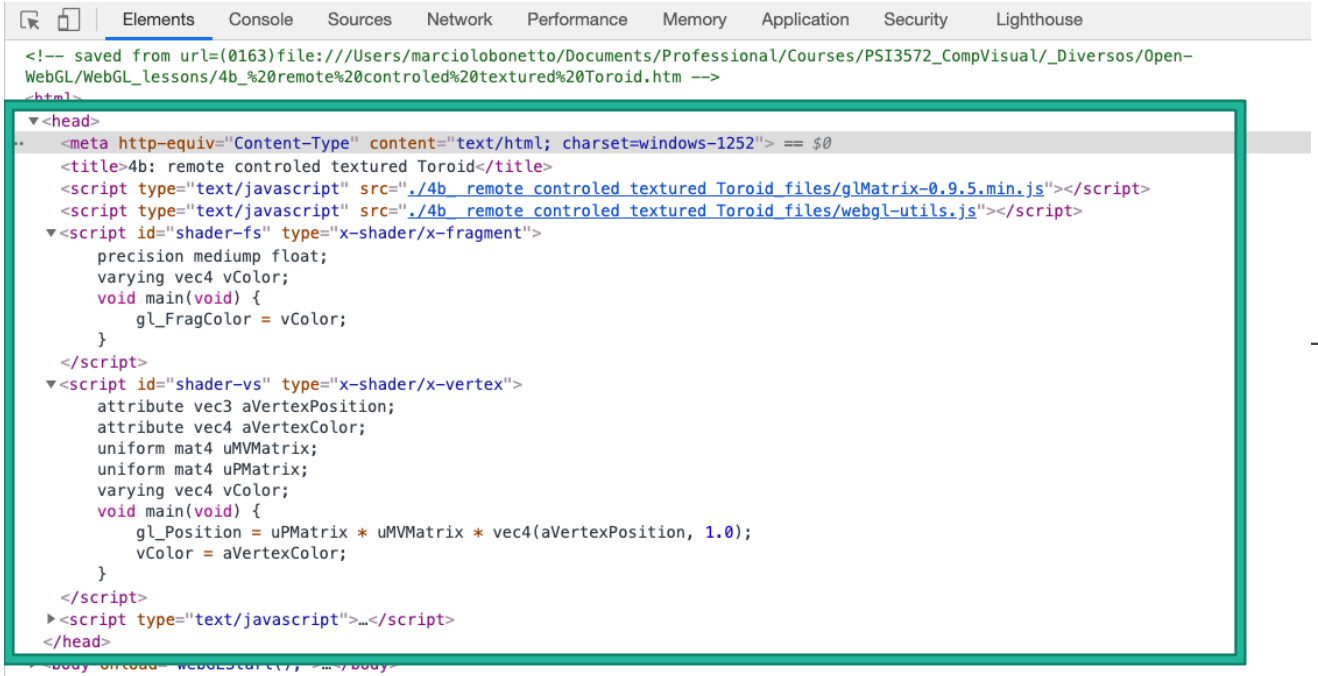

Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
<!-- saved from url=(0163)file:///Users/marciolobonetto/Documents/Professional/Courses/PSI3572_CompVisual/_Diversos/Open-WebGL/WebGL_lessons/4b_%20remote%20controled%20textured%20Toroid.htm -->
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252"> == $0
<title>4b: remote controled textured Toroid</title>
<script type="text/javascript" src="./4b_ remote controled textured Toroid_files/glMatrix-0.9.5.min.js"></script>
<script type="text/javascript" src="./4b_ remote controled textured Toroid_files/webgl-utils.js"></script>
<script id="shader-fs" type="x-shader/x-fragment">…</script>
<script id="shader-vs" type="x-shader/x-vertex">…</script>
<script type="text/javascript">…</script>
<body onload="webGLStart();">
<canvas id="4: remote controled textured Toroid" style="border: none;" width="500" height="500">
<h2>Controls:</h2>

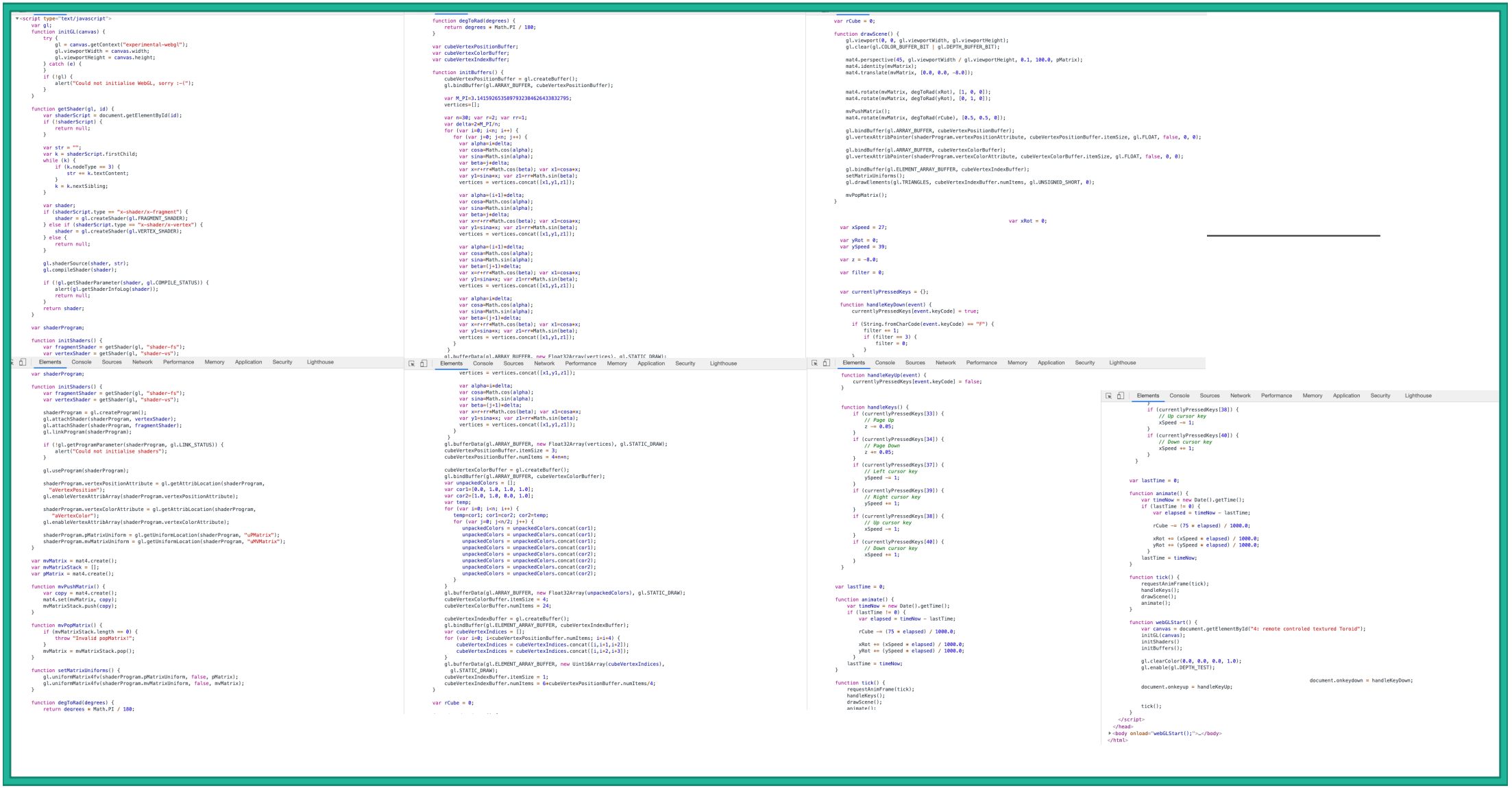
::marker
"Page Up/Page Down to zoom out/in"

::marker
"Cursor keys: make the cube rotate (the longer you hold down a cursor key, the more it accelerates)"

::marker
"F to toggle through three different kinds of texture filters"

</body>
</html>


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
<!-- saved from url=(0163)file:///Users/marciolobonetto/Documents/Professional/Courses/PSI3572_CompVisual/_Diversos/Open-WebGL/WebGL_lessons/4b_%20remote%20controled%20textured%20Toroid.htm -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252"> == $0
<title>4b: remote controled textured Toroid</title>
<script type="text/javascript" src="./4b_ remote controled textured Toroid_files/glMatrix-0.9.5.min.js"></script>
<script type="text/javascript" src="./4b_ remote controled textured Toroid_files/webgl-utils.js"></script>
<script id="shader-fs" type="x-shader/x-fragment">
precision mediump float;
varying vec4 vColor;
void main(void) {
gl_FragColor = vColor;
}
</script>
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
attribute vec4 aVertexColor;
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
varying vec4 vColor;
void main(void) {
gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
vColor = aVertexColor;
}
</script>
<script type="text/javascript">…</script>
</head>
</html>



```
<script type="text/javascript">
    var gl;
    function initGL(canvas) {
        try {
            gl = canvas.getContext("experimental-webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL, sorry :-(");
        }
    }

    function getShader(gl, id) {
        var shaderScript = document.getElementById(id);
        if (!shaderScript) {
            return null;
        }

        var str = "";
        var k = shaderScript.firstChild;
        while (k) {
            if (k.nodeType == 3) {
                str += k.textContent;
            }
            k = k.nextSibling;
        }

        var shader;
        if (shaderScript.type == "x-shader/x-fragment") {
            shader = gl.createShader(gl.FRAGMENT_SHADER);
        } else if (shaderScript.type == "x-shader/x-vertex") {
            shader = gl.createShader(gl.VERTEX_SHADER);
        } else {
            return null;
        }

        gl.shaderSource(shader, str);
        gl.compileShader(shader);

        if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
            alert(gl.getShaderInfoLog(shader));
            return null;
        }
        return shader;
    }

    var shaderProgram;

    function initShaders() {
        var fragmentShader = getShader(gl, "shader-fs");
        var vertexShader = getShader(gl, "shader-vs");
```

```
var shaderProgram;

function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }

    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
      "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram,
      "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
}

var mvMatrix = mat4.create();
var mvMatrixStack = [];
var pMatrix = mat4.create();

function mvPushMatrix() {
    var copy = mat4.create();
    mat4.set(mvMatrix, copy);
    mvMatrixStack.push(copy);
}

function mvPopMatrix() {
    if (mvMatrixStack.length == 0) {
        throw "Invalid popMatrix!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms() {
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}

function degToRad(degrees) {
    return degrees * Math.PI / 180;
}
```

```
function degToRad(degrees) {
    return degrees * Math.PI / 180;
}

var cubeVertexPositionBuffer;
var cubeVertexColorBuffer;
var cubeVertexIndexBuffer;

function initBuffers() {
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);

    var M_PI=3.14159265358979323846264338327795;
    vertices=[];

    var n=30; var r=2; var rr=1;
    var delta=2*M_PI/n;
    for (var i=0; i<n; i++) {
       for (var j=0; j<n; j++) {
          var alpha=i*delta;
          var cosa=Math.cos(alpha);
          var sina=Math.sin(alpha);
          var beta=j*delta;
          var x=r+rr*Math.cos(beta); var x1=cosa*x;
          var y1=sina*x; var z1=rr*Math.sin(beta);
          vertices = vertices.concat([x1,y1,z1]);

          var alpha=(i+1)*delta;
          var cosa=Math.cos(alpha);
          var sina=Math.sin(alpha);
          var beta=j*delta;
          var x=r+rr*Math.cos(beta); var x1=cosa*x;
          var y1=sina*x; var z1=rr*Math.sin(beta);
          vertices = vertices.concat([x1,y1,z1]);

          var alpha=(i+1)*delta;
          var cosa=Math.cos(alpha);
          var sina=Math.sin(alpha);
          var beta=(j+1)*delta;
          var x=r+rr*Math.cos(beta); var x1=cosa*x;
          var y1=sina*x; var z1=rr*Math.sin(beta);
          vertices = vertices.concat([x1,y1,z1]);

          var alpha=i*delta;
          var cosa=Math.cos(alpha);
          var sina=Math.sin(alpha);
          var beta=(j+1)*delta;
          var x=r+rr*Math.cos(beta); var x1=cosa*x;
          var y1=sina*x; var z1=rr*Math.sin(beta);
          vertices = vertices.concat([x1,y1,z1]);
       }
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
```

```
          vertices = vertices.concat([x1,y1,z1]);

          var alpha=i*delta;
          var cosa=Math.cos(alpha);
          var sina=Math.sin(alpha);
          var beta=(j+1)*delta;
          var x=r+rr*Math.cos(beta); var x1=cosa*x;
          var y1=sina*x; var z1=rr*Math.sin(beta);
          vertices = vertices.concat([x1,y1,z1]);
       }
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 4*n*n;

    cubeVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    var unpackedColors = [];
    var cor1=[0.0, 1.0, 1.0, 1.0];
    var cor2=[1.0, 1.0, 0.0, 1.0];
    var temp;
    for (var i=0; i<n; i++) {
       temp=cor1; cor1=cor2; cor2=temp;
       for (var j=0; j<n/2; j++) {
          unpackedColors = unpackedColors.concat(cor1);
          unpackedColors = unpackedColors.concat(cor1);
          unpackedColors = unpackedColors.concat(cor1);
          unpackedColors = unpackedColors.concat(cor1);
          unpackedColors = unpackedColors.concat(cor2);
          unpackedColors = unpackedColors.concat(cor2);
          unpackedColors = unpackedColors.concat(cor2);
          unpackedColors = unpackedColors.concat(cor2);
       }
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
    cubeVertexColorBuffer.itemSize = 4;
    cubeVertexColorBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices = [];
    for (var i=0; i<cubeVertexPositionBuffer.numItems; i=i+4) {
       cubeVertexIndices = cubeVertexIndices.concat([i,i+1,i+2]);
       cubeVertexIndices = cubeVertexIndices.concat([i,i+2,i+3]);
    }
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices),
       gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 6*cubeVertexPositionBuffer.numItems/4;
}

var rCube = 0;
```

```
var rCube = 0;

function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0.0, 0.0, -8.0]);


    mat4.rotate(mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, degToRad(yRot), [0, 1, 0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, degToRad(rCube), [0.5, 0.5, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

    mvPopMatrix();
}


                                                    var xRot = 0;

var xSpeed = 27;

var yRot = 0;
var ySpeed = 39;

var z = -8.0;

var filter = 0;


var currentlyPressedKeys = {};

function handleKeyDown(event) {
    currentlyPressedKeys[event.keyCode] = true;

    if (String.fromCharCode(event.keyCode) == "F") {
        filter += 1;
        if (filter == 3) {
            filter = 0;
        }
    }
```

```
function handleKeyUp(event) {
    currentlyPressedKeys[event.keyCode] = false;
}


function handleKeys() {
    if (currentlyPressedKeys[33]) {
        // Page Up
        z -= 0.05;
    }
    if (currentlyPressedKeys[34]) {
        // Page Down
        z += 0.05;
    }
    if (currentlyPressedKeys[37]) {
        // Left cursor key
        ySpeed -= 1;
    }
    if (currentlyPressedKeys[39]) {
        // Right cursor key
        ySpeed += 1;
    }
    if (currentlyPressedKeys[38]) {
        // Up cursor key
        xSpeed -= 1;
    }
    if (currentlyPressedKeys[40]) {
        // Down cursor key
        xSpeed += 1;
    }
}


var lastTime = 0;

function animate() {
    var timeNow = new Date().getTime();
    if (lastTime != 0) {
        var elapsed = timeNow - lastTime;

        rCube -= (75 * elapsed) / 1000.0;

        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}

function tick() {
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
```

```
        }
        if (currentlyPressedKeys[38]) {
            // Up cursor key
            xSpeed -= 1;
        }
        if (currentlyPressedKeys[40]) {
            // Down cursor key
            xSpeed += 1;
        }
    }


var lastTime = 0;

function animate() {
    var timeNow = new Date().getTime();
    if (lastTime != 0) {
        var elapsed = timeNow - lastTime;

        rCube -= (75 * elapsed) / 1000.0;

        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}

function tick() {
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}

function webGLStart() {
    var canvas = document.getElementById("4: remote controled textured Toroid");
    initGL(canvas);
    initShaders()
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

                                                    document.onkeydown = handleKeyDown;
    document.onkeyup = handleKeyUp;


    tick();
}
</script>
</head>
▶<body onload="webGLStart();">…</body>
</html>
```

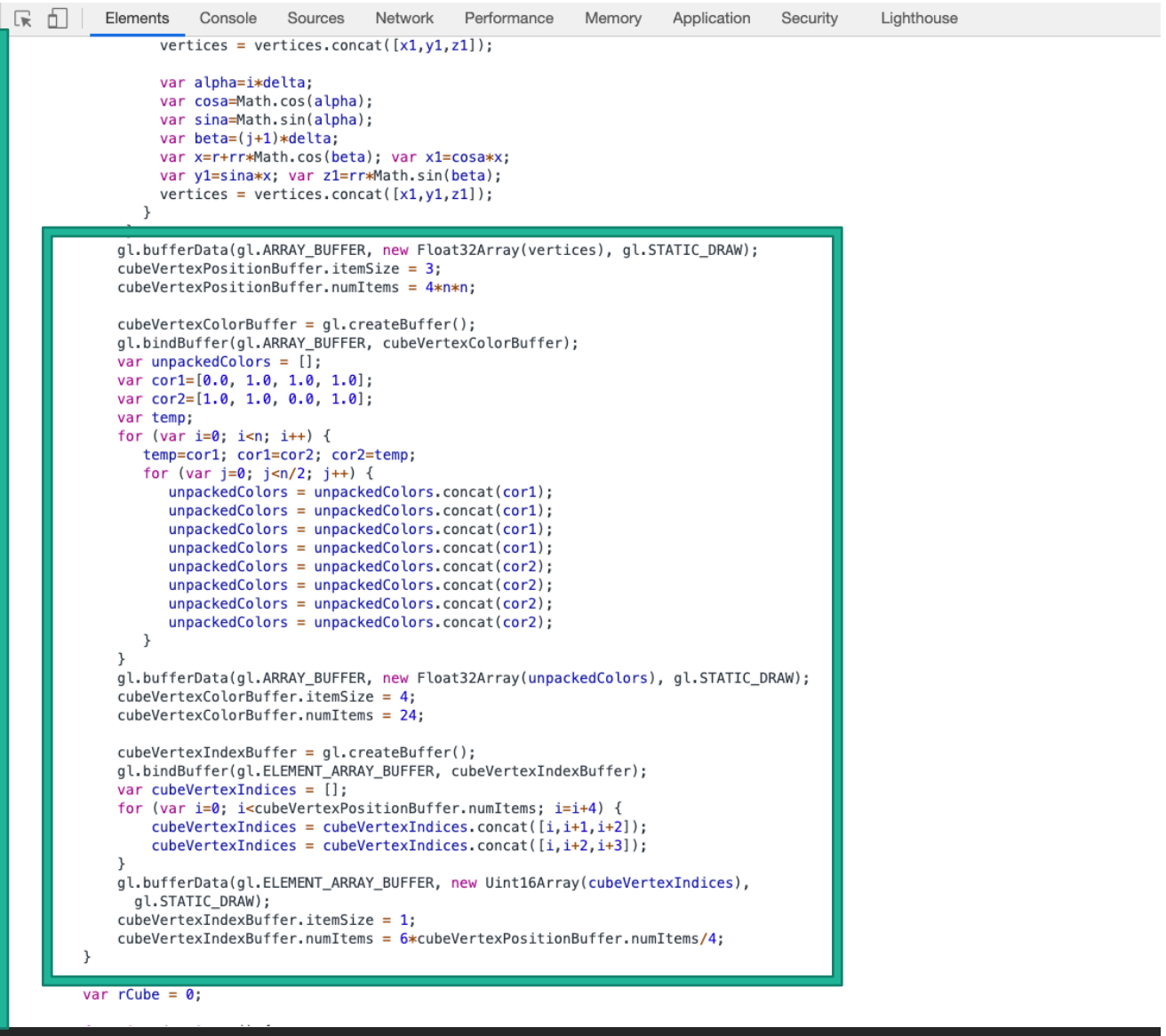
Elements Console Sources Network Performance Memory Application

```
<script type="text/javascript">
    var gl;
    function initGL(canvas) {
        try {
            gl = canvas.getContext("experimental-webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL, sorry :-(");
        }
    }

    function getShader(gl, id) {
        var shaderScript = document.getElementById(id);
        if (!shaderScript) {
            return null;
        }

        var str = "";
        var k = shaderScript.firstChild;
        while (k) {
            if (k.nodeType == 3) {
                str += k.textContent;
            }
            k = k.nextSibling;
        }

        var shader;
        if (shaderScript.type == "x-shader/x-fragment") {
            shader = gl.createShader(gl.FRAGMENT_SHADER);
        } else if (shaderScript.type == "x-shader/x-vertex") {
            shader = gl.createShader(gl.VERTEX_SHADER);
        } else {
            return null;
        }

        gl.shaderSource(shader, str);
        gl.compileShader(shader);

        if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
            alert(gl.getShaderInfoLog(shader));
            return null;
        }
        return shader;
    }

    var shaderProgram;

    function initShaders() {
        var fragmentShader = getShader(gl, "shader-fs");
        var vertexShader = getShader(gl, "shader-vs");
```

Elements Console Sources Network Performance Memory Application Securit Lighthouse

```
var shaderProgram;

function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }

    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
      "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram,
       "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
}

var mvMatrix = mat4.create();
var mvMatrixStack = [];
var pMatrix = mat4.create();

function mvPushMatrix() {
    var copy = mat4.create();
    mat4.set(mvMatrix, copy);
    mvMatrixStack.push(copy);
}

function mvPopMatrix() {
    if (mvMatrixStack.length == 0) {
        throw "Invalid popMatrix!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms() {
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}

function degToRad(degrees) {
    return degrees * Math.PI / 180;
```


Elements Console Sources Network Performance Memory Application
function degToRad(degrees) {
return degrees * Math.PI / 180;
}
var cubeVertexPositionBuffer;
var cubeVertexColorBuffer;
var cubeVertexIndexBuffer;
function initBuffers() {
cubeVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
var M_PI=3.14159265358979323846264338327795;
vertices=[];
var n=30; var r=2; var rr=1;
var delta=2*M_PI/n;
for (var i=0; i<n; i++) {
for (var j=0; j<n; j++) {
var alpha=i*delta;
var cosa=Math.cos(alpha);
var sina=Math.sin(alpha);
var beta=j*delta;
var x=r+rr*Math.cos(beta); var x1=cosa*x;
var y1=sina*x; var z1=rr*Math.sin(beta);
vertices = vertices.concat([x1,y1,z1]);
var alpha=(i+1)*delta;
var cosa=Math.cos(alpha);
var sina=Math.sin(alpha);
var beta=j*delta;
var x=r+rr*Math.cos(beta); var x1=cosa*x;
var y1=sina*x; var z1=rr*Math.sin(beta);
vertices = vertices.concat([x1,y1,z1]);
var alpha=(i+1)*delta;
var cosa=Math.cos(alpha);
var sina=Math.sin(alpha);
var beta=(j+1)*delta;
var x=r+rr*Math.cos(beta); var x1=cosa*x;
var y1=sina*x; var z1=rr*Math.sin(beta);
vertices = vertices.concat([x1,y1,z1]);
var alpha=i*delta;
var cosa=Math.cos(alpha);
var sina=Math.sin(alpha);
var beta=(j+1)*delta;
var x=r+rr*Math.cos(beta); var x1=cosa*x;
var y1=sina*x; var z1=rr*Math.sin(beta);
vertices = vertices.concat([x1,y1,z1]);
}
}


Elements Console Sources Network Performance Memory Application Security Lighthouse
vertices = vertices.concat([x1,y1,z1]);
var alpha=i*delta;
var cosa=Math.cos(alpha);
var sina=Math.sin(alpha);
var beta=(j+1)*delta;
var x=r+rr*Math.cos(beta); var x1=cosa*x;
var y1=sina*x; var z1=rr*Math.sin(beta);
vertices = vertices.concat([x1,y1,z1]);
}
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
cubeVertexPositionBuffer.itemSize = 3;
cubeVertexPositionBuffer.numItems = 4*n*n;
cubeVertexColorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
var unpackedColors = [];
var cor1=[0.0, 1.0, 1.0, 1.0];
var cor2=[1.0, 1.0, 0.0, 1.0];
var temp;
for (var i=0; i<n; i++) {
temp=cor1; cor1=cor2; cor2=temp;
for (var j=0; j<n/2; j++) {
unpackedColors = unpackedColors.concat(cor1);
unpackedColors = unpackedColors.concat(cor1);
unpackedColors = unpackedColors.concat(cor1);
unpackedColors = unpackedColors.concat(cor1);
unpackedColors = unpackedColors.concat(cor2);
unpackedColors = unpackedColors.concat(cor2);
unpackedColors = unpackedColors.concat(cor2);
unpackedColors = unpackedColors.concat(cor2);
}
}
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
cubeVertexColorBuffer.itemSize = 4;
cubeVertexColorBuffer.numItems = 24;
cubeVertexIndexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
var cubeVertexIndices = [];
for (var i=0; i<cubeVertexPositionBuffer.numItems; i=i+4) {
cubeVertexIndices = cubeVertexIndices.concat([i,i+1,i+2]);
cubeVertexIndices = cubeVertexIndices.concat([i,i+2,i+3]);
}
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices),
gl.STATIC_DRAW);
cubeVertexIndexBuffer.itemSize = 1;
cubeVertexIndexBuffer.numItems = 6*cubeVertexPositionBuffer.numItems/4;
}
var rCube = 0;

```
var rCube = 0;

function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0.0, 0.0, -8.0]);


    mat4.rotate(mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, degToRad(yRot), [0, 1, 0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, degToRad(rCube), [0.5, 0.5, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

    mvPopMatrix();
}


                                                    var xRot = 0;
  var xSpeed = 27;

  var yRot = 0;
  var ySpeed = 39;

  var z = -8.0;

  var filter = 0;


  var currentlyPressedKeys = {};

  function handleKeyDown(event) {
      currentlyPressedKeys[event.keyCode] = true;

      if (String.fromCharCode(event.keyCode) == "F") {
          filter += 1;
          if (filter == 3) {
              filter = 0;
          }
      }
```

```
function handleKeyUp(event) {
    currentlyPressedKeys[event.keyCode] = false;
}


function handleKeys() {
    if (currentlyPressedKeys[33]) {
        // Page Up
        z -= 0.05;
    }
    if (currentlyPressedKeys[34]) {
        // Page Down
        z += 0.05;
    }
    if (currentlyPressedKeys[37]) {
        // Left cursor key
        ySpeed -= 1;
    }
    if (currentlyPressedKeys[39]) {
        // Right cursor key
        ySpeed += 1;
    }
    if (currentlyPressedKeys[38]) {
        // Up cursor key
        xSpeed -= 1;
    }
    if (currentlyPressedKeys[40]) {
        // Down cursor key
        xSpeed += 1;
    }
}
```

```
var lastTime = 0;

function animate() {
    var timeNow = new Date().getTime();
    if (lastTime != 0) {
        var elapsed = timeNow - lastTime;

        rCube -= (75 * elapsed) / 1000.0;

        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
      }
    lastTime = timeNow;
}

function tick() {
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}
```

```
        }
        if (currentlyPressedKeys[38]) {
            // Up cursor key
            xSpeed -= 1;
        }
        if (currentlyPressedKeys[40]) {
            // Down cursor key
            xSpeed += 1;
        }
    }
```

```
    var lastTime = 0;

    function animate() {
        var timeNow = new Date().getTime();
        if (lastTime != 0) {
            var elapsed = timeNow - lastTime;

            rCube -= (75 * elapsed) / 1000.0;

            xRot += (xSpeed * elapsed) / 1000.0;
            yRot += (ySpeed * elapsed) / 1000.0;
          }
        lastTime = timeNow;
    }

    function tick() {
        requestAnimFrame(tick);
        handleKeys();
        drawScene();
        animate();
    }

    function webGLStart() {
        var canvas = document.getElementById("4: remote controled textured Toroid");
        initGL(canvas);
        initShaders()
        initBuffers();

        gl.clearColor(0.0, 0.0, 0.0, 1.0);
        gl.enable(gl.DEPTH_TEST);

                                                        document.onkeydown = handleKeyDown;
        document.onkeyup = handleKeyUp;


        tick();
    }
  </script>
 </head>
▶<body onload="webGLStart();">…</body>
</html>
```

Homework

Implemente em WebGL e faça upload do Código no e-disciplinas. Sugestão

- 2 Objetos (cubo, cilindro, piramide, ...)
  animados (girando / se deslocando / pulsando)
  - Sem controle do usuário na animação
  - Com controle do usuário na animação (via painel ou interação com mouse)
  - Sem controle de navegação na cena (observador parado)
  - Com controle de navegação na cenas (via painel ou interação com mouse)
- Numa tela com
  - Painel principal (canvas): cena animada
  - Painel secundário (controle / status): informações e controles da animação e do rendering

Conclusão

Discussão