Teacher: Lý Quốc Ngọc

Lưu Thị Hồng Ngọc

Hồ Thiên Phước

COMPUTER GRAPHIC

HTML5 AND WEBGL

Contents

[1. Comparison OpenGL, WebGL and OpenGL ES 3](#_Toc92659682)

[2. HTML5 3](#_Toc92659683)

[3. INSTALLATION 3](#_Toc92659684)

[4. TRIGGER MECHANISM 4](#_Toc92659685)

[4.1. THREE.JS LIBRARY 4](#_Toc92659686)

[4.2. HOW TO SET UP THREE.JS 5](#_Toc92659687)

[5. 2D DRAWING CONTEXT 5](#_Toc92659688)

[5.1. LINE 5](#_Toc92659689)

[5.2. ARC 5](#_Toc92659690)

[5.3. BIEZER CURVE 5](#_Toc92659691)

[5.4. GEOMETRIC TRANSFORMATION 6](#_Toc92659692)

[5.4.1. TRANSLATE 6](#_Toc92659693)

[5.4.2. ROTATE 6](#_Toc92659694)

[5.4.3. SCALE 6](#_Toc92659695)

[5.5. FILL 7](#_Toc92659696)

[6. 3D DRAWING 7](#_Toc92659697)

[6.1. CREATE AND DISPLAY 3D OBJECT 7](#_Toc92659698)

[6.2. GEOMETRIC TRANSFORMATION 8](#_Toc92659699)

[6.2.1. TRANSLATE 8](#_Toc92659700)

[6.2.2. ROTATE 8](#_Toc92659701)

[6.2.3. SCALE 10](#_Toc92659702)

[6.3. PROJECTION TRANSFORMATION 10](#_Toc92659703)

[6.3.1. PERSPECTIVE PROJECTION 10](#_Toc92659704)

[6.3.2. PARALLEL PROJECTION 11](#_Toc92659705)

[6.4. WIN TO VIEW 12](#_Toc92659706)

[7. GRAPHIC ENVIRONMENT 13](#_Toc92659707)

[HOW TO RENDER WEBGL INTO A PAGE, AN APPLICATION 13](#_Toc92659708)

# Comparison OpenGL, WebGL and OpenGL ES

|  |  |  |  |
| --- | --- | --- | --- |
|  | WebGL | OpenGL | OpenGL ES |
| Definition | Graphics library for web is derived from OpenGL ES (2D and 3D graphics library on embedded systems: motor vehicles, phones, electronics) | Through cross-language and platform API to render 2D and 3D graphic | OpenGL version for embedded systems, typically used on mobile devices |
| Application | Mainly used in browser for web applications | Mainly used in desktop applications | Mainly used in mobile applications |
| Language | Java Script | C, C++, Python, Java | C/C++, Java, Kotlin |
| Feature | Less feature than OpenGL | More feature than WebGL and OpenGL ES | Less feature because it is OpenGL’s embedded  Support other feature better mobile programming |
| Pipeline | No fixed function pipeline | Fixed function pipeline |  |

# HTML5

For applications that need special graphics and motion effects, developers can use Canvas with bitmap style or SVG with vector style. Not only applied to the design of visual web pages, HTML5 is also applied to create graphics libraries that help create graphical applications, games in both 2D and 3D environments such as desktop applications. via JavaScript.

Developers do not need to use or require users to install any libraries in order to run their applications.

# INSTALLATION

Tool: Autodesk Maya, Vizi, …

How to install AutoDesk Maya 2013:

* Download AutoDesk Maya 2013 .exe and run the application
* Exporting the Maya Scene to COLLADA
* Once the exporter is installed, make sure it is turned on in Maya by opening the Plug-in Manager (Window → Settings/Preferences → Plug-in Manager)

Graphical user interface, website

Description automatically generated

* Converting the COLLADA File to glTF. Open Terminal, follows:

|  |
| --- |
| *$ <path-to-converter>/collada2gltf -f futurgo.dae -d*  *[option] export pass details*  *converting:futurgo.dae ... as futurgo.json*  *[shader]: futurgo0VS.glsl*  *[shader]: futurgo0FS.glsl*  *[shader]: futurgo2VS.glsl*  *[shader]: futurgo2FS.glsl*  *[shader]: futurgo4VS.glsl*  *[shader]: futurgo4FS.glsl*  *[completed conversion]* |

After conversion, you will have the file futurgo.json in your folder, along with supporting GLSL shader source files (.glsl file extension). Now that the file has been converted to glTF, we can use the glTF loader for Three.js to load it into the application.

# TRIGGER MECHANISM

## THREE.JS LIBRARY

Three.js is powerful. More than just a wrapper around WebGL, Three.js contains many prebuilt objects useful for developing games, animations, presentations, data visualization, modeling applications, and post-processing special effects. In addition to the capabilities of the core package, there are numerous samples and extras that you can use in your projects.

Three.js is easy to use. The Three.js API has been designed to be friendly and easy to learn. The library comes with many examples that you can use as a starting point.

## HOW TO SET UP THREE.JS

To set up Three.js, go to repository GitHub to download package <https://github.com/mrdoob/three.js/>

# 2D DRAWING CONTEXT

## LINE

|  |
| --- |
| function draw(canvas, context)          {              context.lineTo(x2, y2); //x2, y2 are the last point coordinates  } |

## ARC

|  |
| --- |
| function draw(canvas, context){      context.arc(x, y, radius, startAngle, endAngle, bool counterclockwise);  } |

## BIEZER CURVE

|  |
| --- |
| function draw(canvas, context){  context.bezierCurveTo(x2,y2,x3,y3,x4,y4);  } |

## GEOMETRIC TRANSFORMATION

### TRANSLATE

Text, letter

Description automatically generated

|  |
| --- |
| function draw(canvas, context){  context.translate(x, y);  } |

### ROTATE

Diagram

Description automatically generated

|  |
| --- |
| function draw(canvas, context){              context.rotate(angle);  } |

### SCALE

Text

Description automatically generated

|  |
| --- |
| function draw(canvas, context){                 context.scale(x,y);  } |

## FILL

|  |
| --- |
| function draw(canvas, context){  context.fill();  } |

# 3D DRAWING

## CREATE AND DISPLAY 3D OBJECT

Step 1: Create a data structure including vertices, edges, and polygons. (Example: object specification function)

|  |
| --- |
| var cube = {buffer:vertexBuffer, colorBuffer:colorBuffer, indices:cubeIndexBuffer,  vertSize:3, nVerts:24, colorSize:4, nColors: 24, nIndices:36, primtype:gl.TRIANGLES}; |

Step 2: Determine the projection plane (direction of view, camera coordinates).

|  |
| --- |
| // Add  a camera so we can view the scene  camera = new THREE.PerspectiveCamera( 45,canvas.width / canvas.height, 1, 4000 );  scene.add(camera); |

Step 3: Define projection (parallel, perspective)

Step 4: From real world coordinates, mapping to screen coordinates

Step 5: Draw

## GEOMETRIC TRANSFORMATION

### TRANSLATE

Text

Description automatically generated

|  |
| --- |
| .translate {      -webkit-transform: translateX(20px) translateY(20px) translateZ(-100px);         -moz-transform: translateX(20px) translateY(20px) translateZ(-100px);           -o-transform: translateX(20px) translateY(20px) translateZ(-100px);              transform: translateX(20px) translateY(20px) translateZ(-100px);  } |

### ROTATE

Rotate y axis

A picture containing text

Description automatically generated

|  |
| --- |
| .rotate {          -webkit-transform: rotateY(30deg);             -moz-transform: rotateY(30deg);               -o-transform: rotateY(30deg);                  transform: rotateY(30deg);  } |

Rotate arbitrary rotation axis

Text, letter

Description automatically generated

Diagram

Description automatically generated

Calendar

Description automatically generated

Diagram, text

Description automatically generated

A picture containing text, clock

Description automatically generated

Table

Description automatically generated

### SCALE

Text

Description automatically generated

|  |
| --- |
| .scale {          -webkit-transform: scaleX(1.25) scaleY(.75);             -moz-transform: scaleX(1.25) scaleY(.75);               -o-transform: scaleX(1.25) scaleY(.75);                  transform: scaleX(1.25) scaleY(.75);  } |

## PROJECTION TRANSFORMATION

### PERSPECTIVE PROJECTION

A picture containing calendar

Description automatically generated

Text, letter

Description automatically generated

|  |
| --- |
| .perspective {  -webkit-perspective: 400px;  -moz-perspective: 400px;  -o-perspective: 400px;  perspective: 400px;  } |

### PARALLEL PROJECTION

Text, letter

Description automatically generated

Calendar

Description automatically generated

A picture containing diagram

Description automatically generated

Calendar

Description automatically generated

## WIN TO VIEW

Shape

Description automatically generated with medium confidence

# GRAPHIC ENVIRONMENT

## HOW TO RENDER WEBGL INTO A PAGE, AN APPLICATION

Step 1: Create a Canvas element and obtain a drawing context for the canvas

|  |
| --- |
| function initWebGL(canvas) {          var gl = null;          var msg = "Your browser does not support WebGL, " +              "or it is not enabled by default.";          try          {              gl = canvas.getContext("experimental-webgl");  }  catch (e) {              msg = "Error creating WebGL Context!: " + e.toString();          }  if (!gl) {  alert(msg);              throw new Error(msg);          }  return gl; } |

Step 2: Initialize the viewport

|  |
| --- |
| function initViewport(gl, canvas)      {          gl.viewport(0, 0, canvas.width, canvas.height);      } |

Step 3: Create one or more buffers containing the data to be rendered (typically vertices)

|  |
| --- |
| // Create the vertex data for a square to be drawn  function createSquare(gl) {  var vertexBuffer;  vertexBuffer = gl.createBuffer();  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);  var verts = [  .5, .5, 0.0,  -.5, .5, 0.0,  .5, -.5, 0.0,  -.5, -.5, 0.0  ];  gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(verts), gl.STATIC\_DRAW);  var square = {buffer:vertexBuffer, vertSize:3, nVerts:4,  primtype:gl.TRIANGLE\_STRIP};  return square;  } |

Step 4: Create one or more matrices to define the transformation from vertex buffers to screen space.

|  |
| --- |
| function initMatrices(canvas)      {          // Create a model view matrix with camera at 0, 0, −3.333          var modelViewMatrix = mat4.create();          mat4.translate(modelViewMatrix, modelViewMatrix, [0, 0, −3.333]);          // Create a project matrix with 45 degree field of view          var projectionMatrix = mat4.create();          mat4.perspective(projectionMatrix, Math.PI / 4,              canvas.width / canvas.height, 1, 10000);  } |

Step 5: Create one or more shaders to implement the drawing algorithm

|  |
| --- |
| function createShader(gl, str, type) {          var shader;          if (type == "fragment") {// pixel màu của mỗi đỉnh sau khi biến đổi   shader = gl.createShader(gl.FRAGMENT\_SHADER);          } else if (type == "vertex") {// chuyển toạ độ object thành kgian hiển thị 2D              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;      } |

Step 6: Initialize the shaders with parameters.

|  |
| --- |
| function initShader(gl) {  // load and compile the fragment and vertex shader  var fragmentShader = createShader(gl, fragmentShaderSource,  "fragment");  var vertexShader = createShader(gl, vertexShaderSource,  "vertex");  // link them together into a new program  var shaderProgram = gl.createProgram();  gl.attachShader(shaderProgram, vertexShader);  gl.attachShader(shaderProgram, fragmentShader);  gl.linkProgram(shaderProgram);  // get pointers to the shader params  var shaderVertexPositionAttribute =  gl.getAttribLocation(shaderProgram, "vertexPos");  gl.enableVertexAttribArray(shaderVertexPositionAttribute);  var shaderProjectionMatrixUniform =  gl.getUniformLocation(shaderProgram, "projectionMatrix");  var shaderModelViewMatrixUniform =  gl.getUniformLocation(shaderProgram, "modelViewMatrix");  if (!gl.getProgramParameter(shaderProgram,  gl.LINK\_STATUS)) {  alert("Could not initialise shaders");  } } |

Step 7: Draw

|  |
| --- |
| function draw(gl, obj) {           // clear the background (with black)           gl.clearColor(0.0, 0.0, 0.0, 1.0);           gl.clear(gl.COLOR\_BUFFER\_BIT);           // set the vertex buffer to be drawn           gl.bindBuffer(gl.ARRAY\_BUFFER, obj.buffer);           // set the shader to use           gl.useProgram(shaderProgram);           // connect up the shader parameters: vertex position           // and projection/model matrices           gl.vertexAttribPointer(shaderVertexPositionAttribute,               obj.vertSize, gl.FLOAT, false, 0, 0);           gl.uniformMatrix4fv(shaderProjectionMatrixUniform, false,               projectionMatrix);           gl.uniformMatrix4fv(shaderModelViewMatrixUniform, false,               modelViewMatrix);           // draw the object           gl.drawArrays(obj.primtype, 0, obj.nVerts);        } |