COMPUTER GRAPHIC

HTML5 AND WEBGL

Teacher: Lý Quốc Ngọc
Lưu Thị Hồng Ngọc
Hồ Thiên Phước

Contents

1. 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	

2. 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.

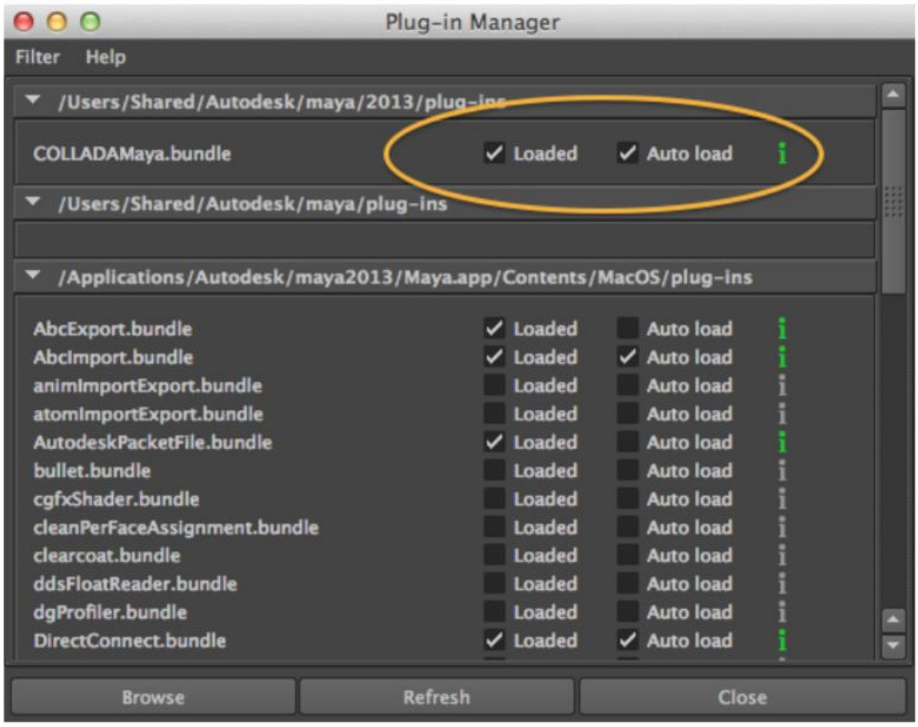
3. INSTALLATION

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)



- 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.

4. TRIGGER MECHANISM

4.1. 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.

4.2. HOW TO SET UP THREE.JS

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

5. 2D DRAWING CONTEXT

5.1. LINE

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

5.2. ARC

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

5.3. BIEZER CURVE

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

5.4. GEOMETRIC TRANSFORMATION

5.4.1. TRANSLATE

$$P(x,y) \rightarrow P'(x',y')$$

$$P' = T(t_x, t_y) \,.\, P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

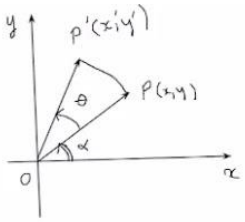
5.4.2. ROTATE

Pivot point: $O(o,o)$

$$P(x,y) \rightarrow P'(x',y')$$

$$P' = R(\theta) \,.\, P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Cos\theta & -Sin\theta & 0 \\ Sin\theta & Cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$




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

5.4.3. SCALE

Pivot point: $O(o,o)$

$$P(x,y) \rightarrow P'(x',y')$$

$$P' = S(O, s_x, s_y) \,.\, P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

5.5. FILL

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

6. 3D DRAWING

6.1. 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

6.2. GEOMETRIC TRANSFORMATION

6.2.1. TRANSLATE

$$P(x,y,z) \rightarrow P'(x',y',z')$$

$$P' = T(t_x,t_y,t_z) \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

```
.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);
}
```

6.2.2. ROTATE

Rotate y axis

Rotate about the y axis.

$$P(x,y,z) \rightarrow P'(x',y',z')$$

$$P' = R(Oy,\theta) \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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

Rotate arbitrary rotation axis

$$R(\vec{V},\theta) = T^{-1}(P_1\vec{O})\,A(\vec{V})^{-1}R(Oz,\theta)A(\vec{V})T(P_1\vec{O})$$

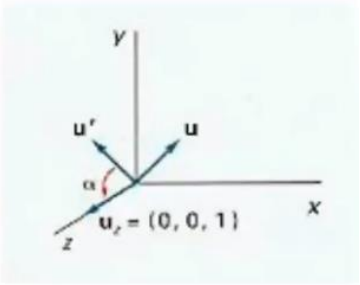
$$A(\vec{V}) = R(Ox,\alpha)R(Oy,\beta)$$

$$A(\vec{V})^{-1} = R(Ox,\alpha)^{-1}R(Oy,\beta)^{-1}$$

$$\cos\alpha = \frac{\vec{u'}.\vec{u_z}}{|u'|.|u_z|} = \frac{c}{d},\quad (\vec{u'} = (0,b,c)), d = \sqrt{b^2+c^2}$$

$$\left.\begin{aligned}\vec{u'}\times\vec{u_z} &= \vec{u_x}.|u'|.|u_z|.\sin\alpha\\ \vec{u'}\times\vec{u_z} &= \vec{u_x}.b\end{aligned}\right\} \Rightarrow \sin\alpha = \frac{b}{d}$$

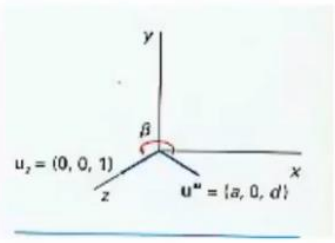
$$R(Ox,\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & c/d & -b/d & 0\\ 0 & b/d & c/d & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$



$$\cos\beta = \frac{\vec{u''}.\vec{u_z}}{|u''|.|u_z|} = d,\quad (\vec{u'} = (a,0,d)), d = \sqrt{b^2+c^2}$$

$$\left.\begin{aligned}\vec{u''}\times\vec{u_z} &= \vec{u_y}.|u''|.|u_z|.\sin\beta\\ \vec{u''}\times\vec{u_z} &= \vec{u_y}.(-a)\end{aligned}\right\} \Rightarrow \sin\beta = -a$$

$$R(Oy,\beta) = \begin{bmatrix} d & 0 & -a & 0\\ 0 & 1 & 0 & 0\\ a & 0 & d & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$



$$R(Oz,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

6.2.3. SCALE

Pivot point: $O(o,o,o)$

$$P(x,y,z)\rightarrow P'(x',y',z')$$

$$P'=S(O,s_x,s_y,s_z).P$$

$$\begin{bmatrix}x'\\ y'\\ z'\\ 1\end{bmatrix}=\begin{bmatrix}s_x & 0 & 0 & 0\\ 0 & s_y & 0 & 0\\ 0 & 0 & s_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

```
.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);
}
```

6.3. PROJECTION TRANSFORMATION

6.3.1. PERSPECTIVE PROJECTION

$$Per(C,(R_0,\vec{N}))=T^{-1}(\vec{CO}).Per(O,(R_0,\vec{N})).T(\vec{CO})$$

$$=\begin{bmatrix}1 & 0 & 0 & c_x\\ 0 & 1 & 0 & c_y\\ 0 & 0 & 1 & c_z\\ 0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}d & 0 & 0 & 0\\ 0 & d & 1 & 0\\ 0 & 0 & d & 0\\ n_1 & n_2 & n_3 & 0\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & 0 & -c_x\\ 0 & 1 & 0 & -c_y\\ 0 & 0 & 1 & -c_z\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Projection Reference Point: $C(c_x, c_y, c_z)$

View Plane: $(R_0, \vec{N}), R_0(x_0, y_0, z_0), \vec{N}(n_1, n_2, n_3)$

$$P' = Per(C, (R_0, \vec{N})).P$$

$$\begin{bmatrix} x_H \\ y_H \\ z_H \\ H \end{bmatrix} = \begin{bmatrix} d + c_x.n_1 & c_x.n_2 & c_x.n_3 & -c_x.d_0 \\ c_y.n_1 & d + c_y.n_2 & c_y.n_3 & -c_y.d_0 \\ c_z.n_1 & c_z.n_2 & d + c_z.n_3 & -c_z.d_0 \\ n_1 & n_2 & n_3 & -d_1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$d_0 = n_1 x_0 + n_2 y_0 + n_3 z_0 \quad d_0 = n_1.c_x + n_2.c_y + n_3.c_z$$

$$d = d_0 - d_1$$

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

6.3.2. PARALLEL PROJECTION

Projection Line: $\vec{V}(a, b, c)$

View Plane: $(R_0, \vec{N})$

$$P' = Par(\vec{V}, (R_0, \vec{N})).P$$

$$Par(\vec{V}, (R_0, \vec{N})) = T(\vec{OR_0}).A^{-1}(\vec{N}).Par(\vec{V}, (O, Oz).A(\vec{N}).T(\vec{R_0O})$$

$$Par(\vec{V}, (O, Oz) = \begin{bmatrix} 1 & 0 & -\frac{a}{c} & 0 \\ 0 & 1 & -\frac{b}{c} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A(\vec{N}) = \begin{bmatrix} \frac{\lambda}{|\vec{N}|} & \frac{-n_1 n_2}{\lambda |\vec{N}|} & \frac{-n_1 n_3}{\lambda |\vec{N}|} & 0 \\ 0 & \frac{n_3}{\lambda} & -\frac{n_2}{\lambda} & 0 \\ \frac{n_1}{|\vec{N}|} & \frac{n_2}{|\vec{N}|} & \frac{n_3}{|\vec{N}|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \lambda = \sqrt{n_2^2 + n_3^2}$$

$$T(\vec{R_0 O}) = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.4. WIN TO VIEW

$$P(x, y) \rightarrow P'(x', y')$$

$$P' = WTV.P$$

$$s_x = \frac{v_{max}.x - v_{min}.x}{w_{max}.x - w_{min}.x}$$

$$s_y = \frac{v_{max}.y - v_{min}.y}{w_{max}.y - w_{min}.y}$$

$$t_x = v_{min}.x - w_{min}.x$$

$$t_y = v_{min}.y - w_{min}.y \Rightarrow WTV = S(v_{min}, s_x, s_y).T(t_x, t_y)$$

$$\Rightarrow WTV = \begin{pmatrix} s_x & 0 & v_{min}.x - s_x.v_{min}.x \\ 0 & s_y & v_{min}.y - s_y.v_{min}.y \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 1 & 0 & v_{min}.x - w_{min}.x \\ 0 & 1 & v_{min}.y - w_{min}.y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} s_x & 0 & v_{min}.x - s_x.w_{min}.x \\ 0 & s_y & v_{min}.y - s_y.w_{min}.y \\ 0 & 0 & 1 \end{pmatrix}$$

7. 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);
    }
```