Transformation, perspective projection, and LookAT in WebGL vs. OpenGL

Ming Ouhyoung, September 2018
To make things (functions) simple:

• WebGL is an Open ES 2.0 binding. OpenGL ES 2.0 (and modern OpenGL 3.2+) does not have some old OpenGL functions, everything must be done in shaders and or your own matrix libraries.

• Good thing is that there is plenty of matrix libraries available for WebGL, one of the best/fastest being glMatrix (https://github.com/toji/gl-matrix).
• But where are glLoadIdentity, glMultMatrix, glTranslate, and glRotate, in OpenGL?
Library: glMatrix

- Javascript has evolved into a language capable of handling realtime 3D graphics, via WebGL, and computationally intensive tasks such as physics simulations. These types of applications demand high performance vector and matrix math, which is something that Javascript doesn't provide by default. glMatrix to the rescue!
• glMatrix is designed to perform vector and matrix operations stupidly fast! By hand-tuning each function for maximum performance and encouraging efficient usage patterns through API conventions, glMatrix will help you get the most out of your browsers Javascript engine.
glMatrix API

• The *glMatrix* API can be made available for use on a web page with a script element such as

• `<script src="gl-matrix-min.js"></script>` This assumes that *gl-matrix-min.js* is in the same directory as the web page.
Each *glMatrix* class has a *create()* function which creates an array of the appropriate length and fills it with default values. For example,

- `transform = mat4.create();` sets *transform* to be a new *Float32Array* of length 16, initialized to represent the identity matrix. Similarly,
- `vector = vec3.create();` creates a *Float32Array* of length 3, filled with zeros. Each class also has a function *clone(x)* that creates a copy of its parameter *x*. For example:
  - `saveTransform = mat4.clone(modelview);`
Translation, rotation

To apply a translation by a vector \([dx,dy,dz]\), we can say

- \texttt{mat4.translate( modelview, modelview, [dx,dy,dz] )}; // This is equivalent to calling \texttt{glTranslatef(dx,dy,dz)} in OpenGL.

To apply a scaling transformation with scale factors \(sx, sy, \text{ and } sz\), use

- \texttt{mat4.scale( modelview, modelview, [sx,sy,sz] )};

in OpenGL: \texttt{glScalef(sx, sy, sz)}
glMatrix for rotation: These functions allow us to do all the basic modeling and viewing transformations that we need for 3D graphics.

For rotation, glMatrix has four functions, including three for the common cases of rotation about the x, y, or z axis. The fourth rotation function specifies the axis of rotation as the line from (0,0,0) to a point (dx,dy,dz). This is equivalent to glRotatef(angle,dx,dy,dz).

Unfortunately, the angle of rotation in these functions is specified in radians rather than in degrees:

- mat4.rotateX(modelview, modelview, radians);
- mat4.rotateY(modelview, modelview, radians);
- mat4.rotateZ(modelview, modelview, radians);
- mat4.rotate(modelview, modelview, radians, [dx,dy,dz]);
mat4.rotate(out, a, rad, axis)

Rotates a mat4 by the given angle around the given axis

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mat4</td>
<td>the receiving matrix</td>
</tr>
<tr>
<td>a</td>
<td>mat4</td>
<td>the matrix to rotate</td>
</tr>
<tr>
<td>rad</td>
<td>Number</td>
<td>the angle to rotate the matrix by</td>
</tr>
<tr>
<td>axis</td>
<td>vec3</td>
<td>the axis to rotate around</td>
</tr>
</tbody>
</table>
mat4.scale(out, a, v)

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mat4</td>
<td>the receiving matrix</td>
</tr>
<tr>
<td>a</td>
<td>mat4</td>
<td>the matrix to scale</td>
</tr>
<tr>
<td>v</td>
<td>vec3</td>
<td>the vec3 to scale the matrix by</td>
</tr>
</tbody>
</table>
The *glMatrix* library has a "lookAt" function to do the same thing:

```javascript
mat4.lookAt( modelview, [eyex,eyey,eyez], [refx,refy,refz], [upx,upy,upz] );
```

- var modelview = mat4.create();//create
- mat4.identity( modelview ); //set to identity
- This function call is actually equivalent to the two OpenGL commands
  ```javascript
  glLoadIdentity();
  gluLookAt( eyex,eyey,eyez,refx,refy,refz,upx,upy,upz );
  ```
Perspective Projection

- `mat4.ortho(projection, left, right, bottom, top, near, far);`
- `mat4.frustum(projection, left, right, bottom, top, near, far);`
- `mat4.perspective(projection, fovyInRadians, aspect, near, far);`

As with the modelview transformation, you do not need to load `projection` with the identity before calling one of these functions, but you must create `projection` as a `mat4` (or an array of length 16).
Other original WebGL functions: 1

• `gl.TRIANGLES`
  To draw a series of separate triangles.

        `gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0);`

• `gl.TRIANGLE_STRIP`

• To draw a series of connected triangles in strip fashion.
• **gl.POINTS**
To draw a series of points.

```javascript
gl.drawArrays(gl.POINTS, 0, 3);  //draw three points.
```

• **gl.LINES**
To draw a series of unconnected line segments (individual lines).

```javascript
gl.drawArrays(gl.LINES, 0, 2);  //Every line needs two indices: starting point and end point:
```

• **gl.LINE_STRIP**
To draw a series of connected line segments.
void gl.drawElements(mode, count, type, offset);

- gl.POINTS: Draws a single dot.
- gl.LINE_STRIP: Draws a straight line to the next vertex.
- gl.LINE_LOOP: Draws a straight line to the next vertex, and connects the last vertex back to the first.
- gl.LINES: Draws a line between a pair of vertices.
- gl.TRIANGLE_STRIP
- gl.TRIANGLE_FAN
- gl.TRIANGLES: Draws a triangle for a group of three vertices.
Example:

• `gl.drawElements(gl.POINTS, 8, gl.UNSIGNED_BYTE, 0);`
Define the element array

- const indexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);

  // This array defines each face as two triangles, using the indices into the vertex array to specify each triangle's position.
  const indices = [ 0, 1, 2, 0, 2, 3, front 4, 5, 6, 4, 6, 7, back 8, 9, 10, 8, 10, 11, top 12, 13, 14, 12, 14, 15, bottom 16, 17, 18, 16, 18, 19, right 20, 21, 22, 20, 22, 23, left ];

  // Now send the element array to GL
Other original WebGL functions: 2

- `gl.enable(gl.DEPTH_TEST);`
  `gl.depthFunc(gl.LEQUAL);`
- `gl clearColor(0.5, 0.5, 0.5, 0.9);`
  `gl.clearDepth(1.0);`
- `gl.viewport(0.0, 0.0, canvas.width, canvas.height);`
- `gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);`