WebGL Program

WebGL Program -- Five Steps

- Describe page (HTML file)
  - request WebGL Canvas
  - read in necessary files
- Define shaders (HTML file)
- Compute or specify data (JS file)
- Send data to GPU (JS file)
- Render data (JS file)

Square.html

```
<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
void main(){
    gl_Position = vPosition;
}
</script>
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main()
{
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>
```

Shaders

- Assign names to the shaders to be used in the JS file
- These are trivial pass-through shaders that which set the two required built-in variables
  - gl_Position
  - gl_FragColor
- Note both shaders are full programs
- Must set precision in fragment shader

More Square.html

```
<script type="text/javascript" src="../Common/webgl-
utils.js"></script>
<script type="text/javascript" src="../Common/
initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="square.js"></script>
</head>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

Files

- **`../Common/webgl-utils.js`**: Standard utilities for setting up WebGL context in Common directory on website
- **`../Common/initShaders.js`**: contains JS and WebGL code for reading, compiling and linking the shaders
- **`../Common/MV.js`**: our matrix-vector package
- **`square.js`**: the application file

square.js

```
//  Configure WebGL
 gl.viewport( 0, 0, canvas.width, canvas.height );
 gl.clearColor( 0.0, 0.0, 0.0, 1.0 );
//  Load shaders and initialize attribute buffers
 var program = initShaders( gl, "vertex-shader", "fragment-
shader" );
 gl.useProgram( program );
  // Load the data into the GPU
 var bufferId = gl.createBuffer();
 gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
 gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices),
gl.STATIC_DRAW );
// Associate out shader variables with our data buffer
 var vPosition = gl.getAttribLocation( program, "vPosition" );
 gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0,
0 );
 gl.enableVertexAttribArray( vPosition );
```

Notes

- **`initShaders`** used to load, compile and link shaders to form a program object
- Load data onto GPU by creating a **vertex buffer object** on the GPU
  - Note use of flatten() to convert JS array to an array of float32's
- Finally we must connect variable in program with variable in shader
  - need name, type, location in buffer

Program Execution

- WebGL runs within the browser
  - complex interaction among the operating system, the window system, the browser and your code (HTML and JS)
- Simple model
  - Start with HTML file
  - start with onload function
    - event driven input

Coordinate Systems

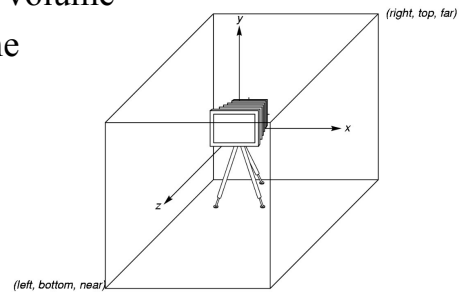
- The units in **`points`** are determined by the application and are called *object, world, model* or *problem coordinates*
- Viewing specifications usually are also in object coordinates
- Eventually pixels will be produced in *window coordinates*
- *clip coordinates*

Coordinate Systems and Shaders

- Vertex shader must output in clip coordinates
- Input to fragment shader from rasterizer is in window coordinates
- Application can provide vertex data in any coordinate system but shader must eventually produce gl_Position in clip coordinates
- Simple example uses clip coordinates

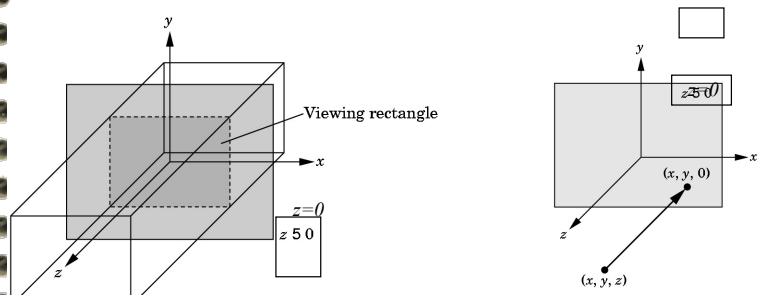
WebGL Camera

- WebGL places a camera at the origin in object space pointing in the negative *z* direction
- The default viewing volume is a box centered at the origin with sides of length 2



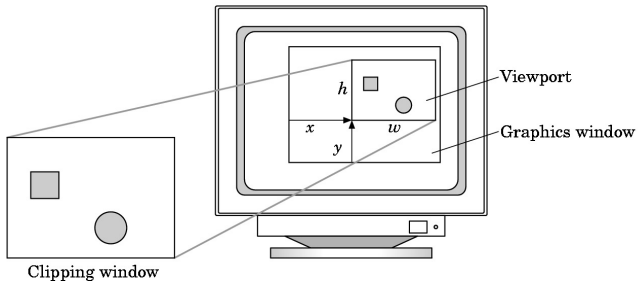
Orthographic Viewing

In the default orthographic view, points are projected forward along the z axis onto the plane $z=0$



Viewports

- Do not have use the entire window for the image: **`gl.viewport(x,y,w,h)`**
- Values in pixels (window coordinates)



Transformations and Viewing

- In WebGL, we usually carry out projection using a projection matrix (transformation) before rasterization
- Transformation functions are also used for changes in coordinate systems
- Three choices in WebGL
  - Application code
  - GLSL functions
  - MV.js