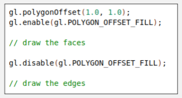
Lab 3

- `mat4.frustum` vs `mat4.perspective`
  - both specify a perspective projection – the difference is just convenience
    - it can be easier to think in terms of the view window rather than a field of view angle
      - `mat4.frustum(A,left,right,bottom,top,near,far)`
      - `mat4.perspective(A,fieldOfView,aspect,near,far)`
        - `fieldOfView` in radians
    - `fieldOfView` is in radians, not degrees

Lab 3

- drawing
  - for wireframe – need to draw one LINE_LOOP per face
    - LINE_LOOP treats all of the vertices as a single polygon
  - to ensure that wireframe is visible over solid polygons, need to draw the lines offset a bit from the polygons

- problem: pixels along polygon edges are at the same depth whether drawing faces (filled polygons) or edges (wireframe)



```
gl.polygonOffset(1.0, 1.0);
gl.enable(gl.POLYGON_OFFSET_FILL);

// draw the faces

gl.disable(gl.POLYGON_OFFSET_FILL);

// draw the edges
```

the solution is to tell OpenGL to draw the filled polygons slightly offset in depth from the wireframe

`gl.polygonOffset(`*`factor`*`,`*`units`*`)`
– `factor` allows for different offsets depending on the angle of the polygon into the screen – 1 is generally fine
– `units` specifies the size of the offset

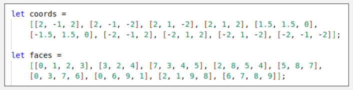
this offsets the filled polygons; you can instead offset the wireframe edges

Lab 3

- repeated vertices in an indexed face set representation
  - appropriate for polyhedron to be able to use polygon normals for flat shading, but otherwise the point is *not* to repeat vertices

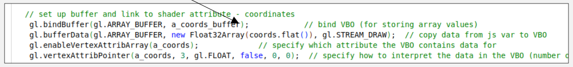
Lab 3

- specifying geometry
  - when sending values to the shader, you need a 1D array (`Float32Array` or similar) – but you don't have to start with that
  - an array-of-arrays is convenient
    - can then programmatically build the `Float32Array`



```
let coords =
    [[2, -1, 2], [2, -1, -2], [2, 1, -2], [2, 1, 2], [1.5, 1.5, 0],
    [-1.5, 1.5, 0], [-2, -1, 2], [-2, 1, 2], [-2, 1, -2], [-2, -1, -2]];

let faces =
    [[0, 1, 2, 3], [3, 2, 4], [7, 3, 4, 5], [2, 8, 5, 4], [5, 8, 7],
    [0, 3, 7, 6], [0, 6, 9, 1], [2, 1, 9, 8], [6, 7, 8, 9]];
```



```
  // set up buffer and link to shader attribute - coordinates
   gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);          // bind VBO (for storing array values)
   gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(coords.flat()), gl.STREAM_DRAW);  // copy data from js var to VBO
   gl.enableVertexAttribArray(a_coords);            // specify which attribute the VBO contains data for
   gl.vertexAttribPointer(a_coords, 3, gl.FLOAT, false, 0, 0);  // specify how to interpret the data in the VBO (number o
```

  - house

`models-IFS.js` and `teapot-model-IFS.js`, the house on page 5 of Monday's "specifying geometry" slides, and one object of your own where the