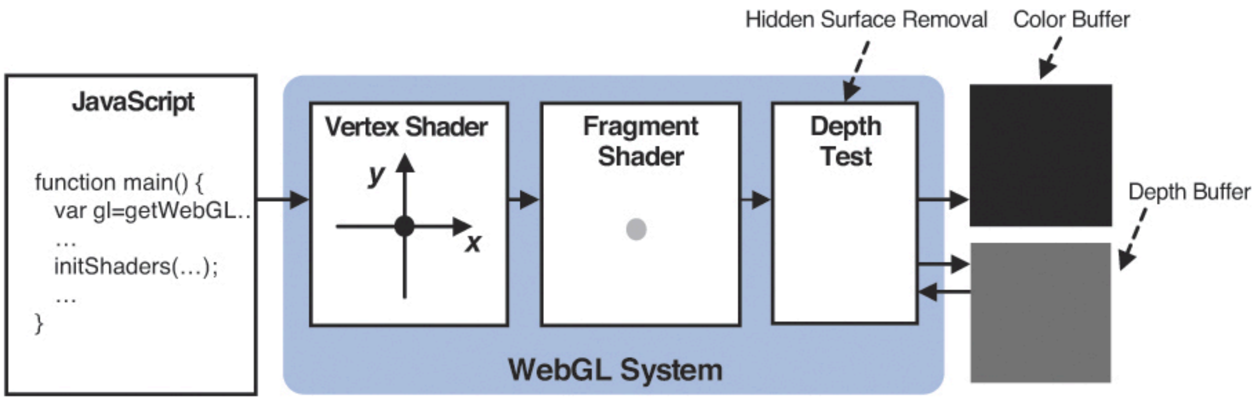
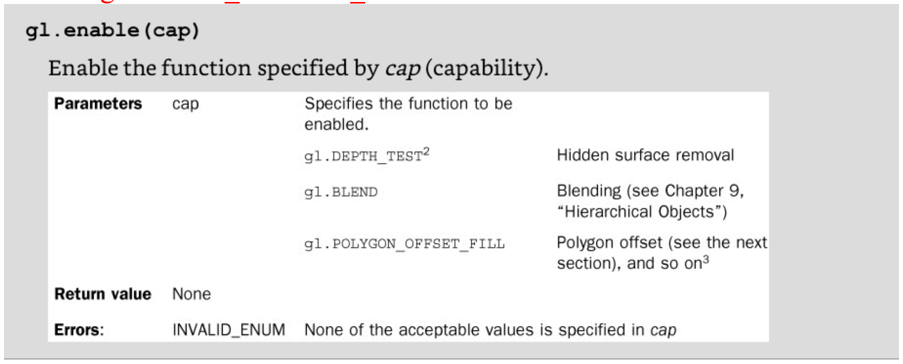
Drawing 3D Objects

- **Draw a cone**

- **Depth buffer and hidden surface removal**



  - In order to perform hidden surface removal, the current frame buffer must have a depth buffer. A depth buffer is an image that uses a depth image format.
  - Fragment depth value: Every fragment has a depth value. This value is either computed by the fragment shader, or is the window-space z coordinate computed as the output of the vertex post-processing steps.
  - Depth test is a per-sample processing operation performed after the Fragment Shader (and sometimes before). The Fragment's output depth value may be tested against the depth of the sample being written to. If the test fails, the fragment is discarded. If the test passes, the depth buffer will be updated with the fragment's output depth, unless a subsequent per-sample operation prevents it.
  - How to use the depth buffer? Two steps:
    1. Enable the hidden surface removal function: gl.enable(gl.DEPTH_TEST). (This function can be turned off with: gl.disable(gl.DEPTH_TEST)
    2. Clear the depth buffer used for the hidden surface removal before drawing: gl.DEPTH_BUFFER_BIT



gl.enable(cap)

Enable the function specified by *cap* (capability).

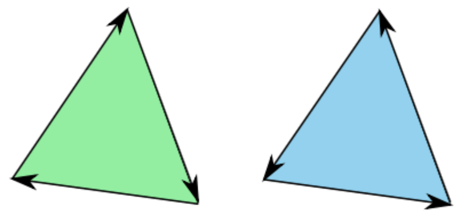
Parameters	cap	Specifies the function to be enabled.	
		`gl.DEPTH_TEST`[2]	Hidden surface removal
		`gl.BLEND`	Blending (see Chapter 9, "Hierarchical Objects")
		`gl.POLYGON_OFFSET_FILL`	Polygon offset (see the next section), and so on[3]
Return value	None		
Errors:	INVALID_ENUM	None of the acceptable values is specified in *cap*	

- **Winding order**

  Triangle primitives after all transformation steps have a particular facing. Triangle that faces the viewer is called "Front face", triangle not facing the viewer is called "Back face". This is defined by the order of the three vertices that make up the triangle, as well as their apparent order on-screen. For efficiency purposes, triangles can be discarded, i.e., not drawn, based on their apparent facing, a process known as **Face Culling**.

  When vertices are broken down into Primitives during Primitive Assembly, the order of the vertices relative to the others in the primitive is noted. The order of the vertices in a triangle, when combined with their visual orientation, can be used to determine whether the triangle is being seen from the "front" or the "back" side.

This is determined by the winding order of the triangle. Given an ordering of the triangle's three vertices, a triangle can appear to have a clockwise winding or counter-clockwise winding. Clockwise means that the three vertices, in order, rotate clockwise around the triangle's center. Counter-clockwise means that the three vertices, in order, rotate counter-clockwise around the triangle's center.

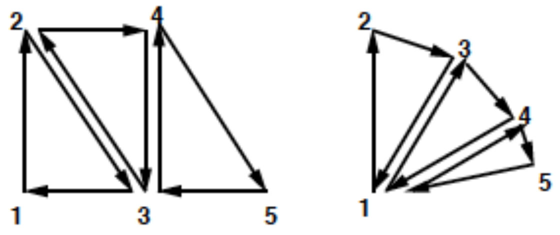


clockwise
back facing

counter clockwise
front facing (default)

(winding maybe changed: gl.frontFace(gl.CW) ➔ making clockwise the front face)

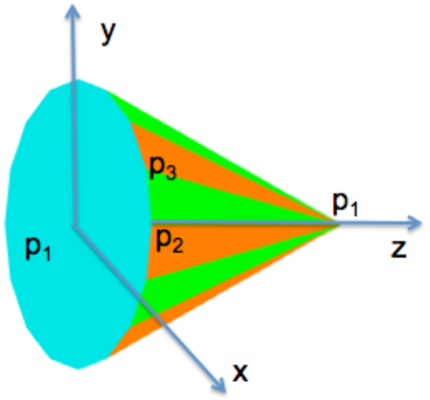
- What about triangle strip and triangle fan?



triangle strip

triangle fan

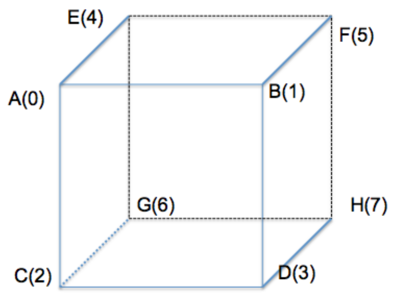
- In 3Dtriangle.js, which face is front facing and which face is back facing?



- Cull face – selectively not to render either the front or back faces of a triangle
  The option is turned off by default. May use enable(gl.CULL_FACE), disable(gl.CULL_FACE) to turn it on or off. The options are: gl.FRONT, gl.BACK, gl.FRONT_AND_BACK

- **Drawing a cube**
    - Approach one: Use quad to define in .js file

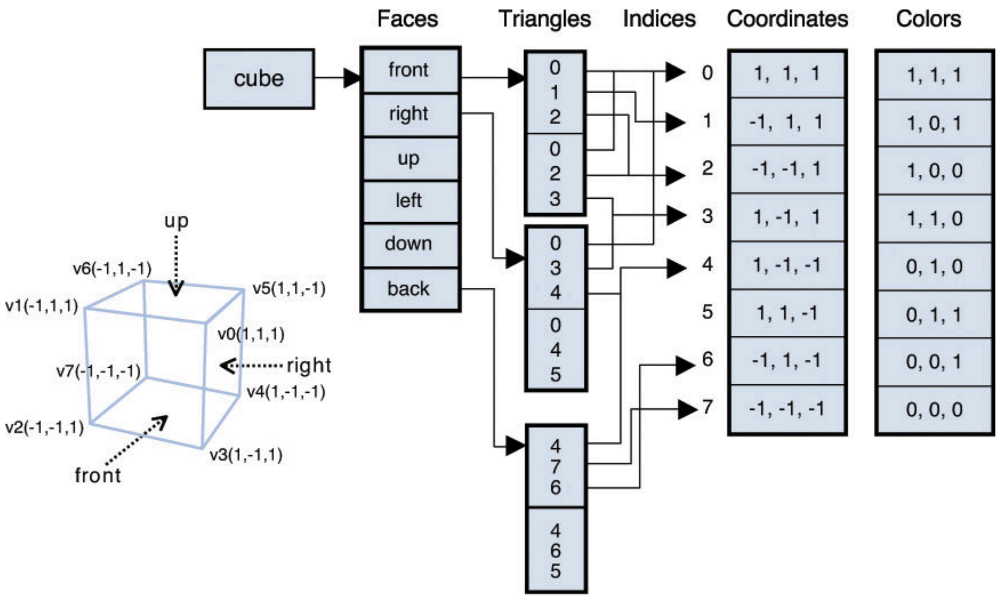
```
var vertices = [
        vec4(-1,  1,  1, 1.0 ),  // A(0)
        vec4( 1,  1,  1, 1.0 ),  // B (1)
        vec4(-1, -1,  1, 1.0 ),  // C (2)
        vec4( 1, -1,  1, 1.0 ), // D (3)
        vec4( -1, 1, -1, 1.0 ), // E (4)
        vec4( 1,  1, -1, 1.0 ), // F(5)
        vec4( -1,-1, -1, 1.0 ), // G(6
        vec4( 1, -1, -1, 1.0 ),  // H(7)
    ];


var vertexColors = [
        vec4( 1.0, 1.0, 1.0, 1.0 ),  // white
        vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
        vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
        vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
        vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
        vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
        vec4( 0.0, 1.0, 1.0, 1.0 ),  // cyan
    ];

function colorCube() {
    quad( 1, 3, 2, 0 );  // front red
    quad( 4, 5, 7, 6 );  // back blue
    quad( 3, 1, 5, 7 );  // right green
    quad( 6, 2, 0, 4 );  // left cyan
    quad( 2, 3, 7, 6 ); // bottom yellow
    quad( 0, 1, 5, 4); // top white
}

function quad(a, b, c, d) {  // a is color index
    pointsArray.push(vertices[a]);
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[b]);        ← first triangle
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[c]);
    colorsArray.push(vertexColors[a]);

    pointsArray.push(vertices[a]);
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[c]);        ← second triangle
    colorsArray.push(vertexColors[a]);
    pointsArray.push(vertices[d]);
    colorsArray.push(vertexColors[a]);   }
```



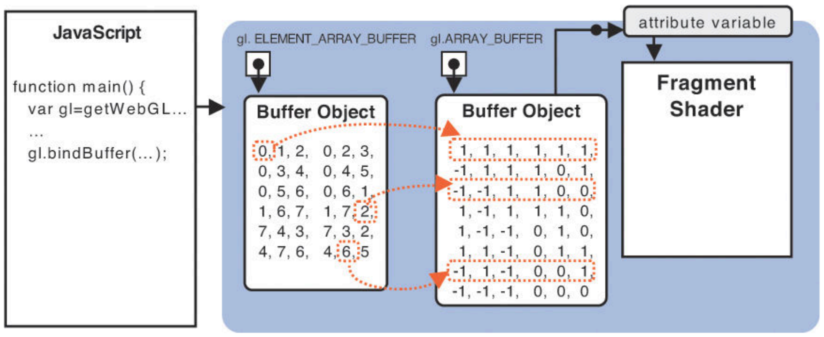
        - Are all the faces defined in the example code front facing? (if front facing is defined as CCW)
        - How to create a square table top? the pillars to a building? a stick? the sides of a drawer? a door? the particles? …

What is a disadvantage of the above representation method?

- **Approach two: use element index buffer** (ch04/cubev.js)
  gl.ELEMENT_ARRAY_BUFFER

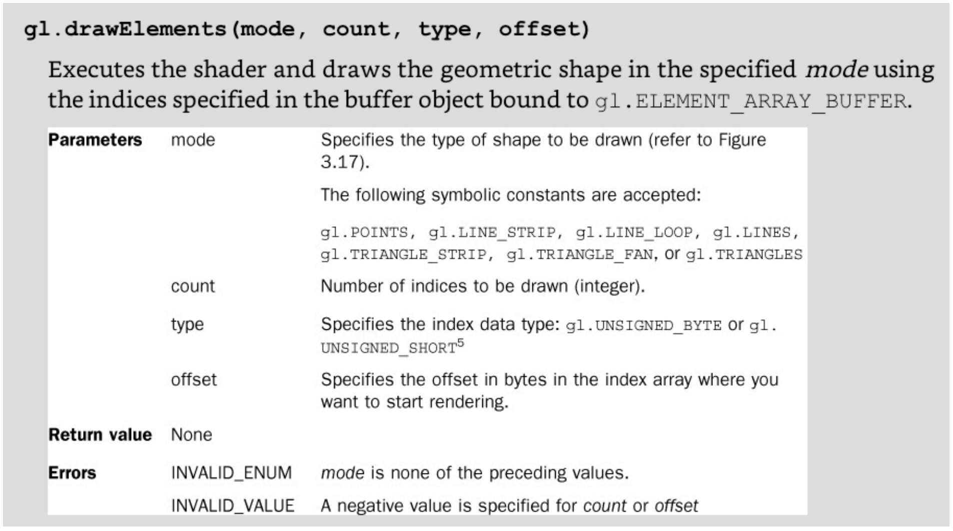


Indices	Coordinates	Colors
0	1, 1, 1	1, 1, 1
1	-1, 1, 1	1, 0, 1
2	-1, -1, 1	1, 0, 0
3	1, -1, 1	1, 1, 0
4	1, -1, -1	0, 1, 0
5	1, 1, -1	0, 1, 1
6	-1, 1, -1	0, 0, 1
7	-1, -1, -1	0, 0, 0



```
var indices = [ 0, 1, 2,          0, 2, 3,   // front
                 0, 3, 4,          0, 4, 5,  // right
                0, 5, 6,           0,  6, 1,   // up
                1, 6, 7,           1, 7, 2,  // left
                7, 4, 3,           7, 3, 2,   // down
                4, 7, 6,           4, 6, 5];  // back
```

```
// array element buffer
var iBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, iBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(indices), gl.STATIC_DRAW);
```



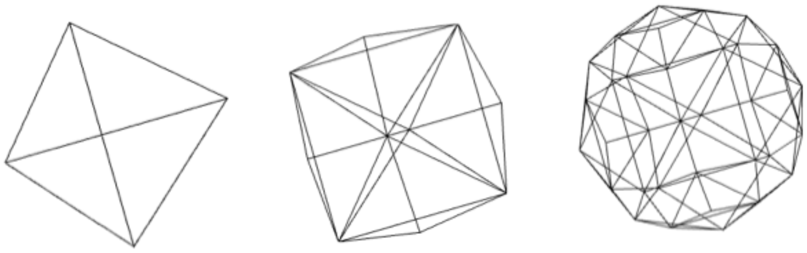
gl.drawElements(mode, count, type, offset)

Executes the shader and draws the geometric shape in the specified *mode* using the indices specified in the buffer object bound to `gl.ELEMENT_ARRAY_BUFFER`.

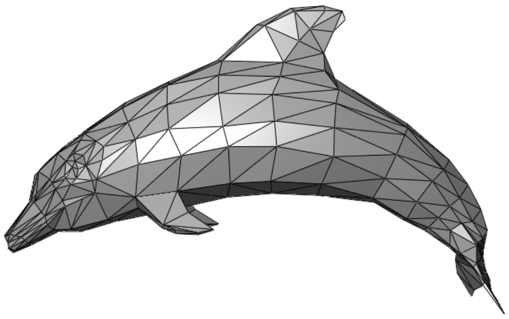
Parameters	mode	Specifies the type of shape to be drawn (refer to Figure 3.17). The following symbolic constants are accepted: `gl.POINTS`, `gl.LINE_STRIP`, `gl.LINE_LOOP`, `gl.LINES`, `gl.TRIANGLE_STRIP`, `gl.TRIANGLE_FAN`, or `gl.TRIANGLES`
	count	Number of indices to be drawn (integer).
	type	Specifies the index data type: `gl.UNSIGNED_BYTE` or `gl.UNSIGNED_SHORT`[5]
	offset	Specifies the offset in bytes in the index array where you want to start rendering.
Return value	None	
Errors	INVALID_ENUM	*mode* is none of the preceding values.
	INVALID_VALUE	A negative value is specified for *count* or *offset*

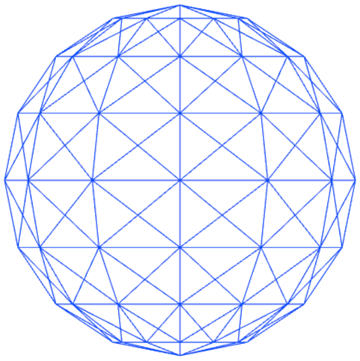
```
gl.drawElements( gl.TRIANGLES, numVertices, gl.UNSIGNED_BYTE, 0 );
```

- **Draw a sphere**
  - Two ways of drawing a sphere
    - recursive subdivision (ch06/wireSphere.js)
      - how many triangles are there for each version?
      - how does the subdivision go from one to the next version?

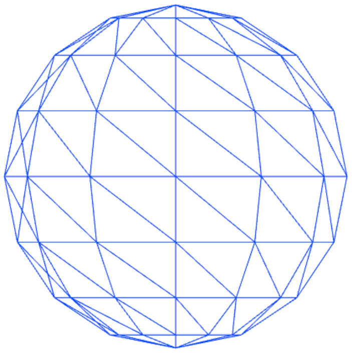


    - mesh object comprises a set of triangles (typically in three dimensions) that are connected by their common edges or corners.





Full Sphere



Front half of the Sphere