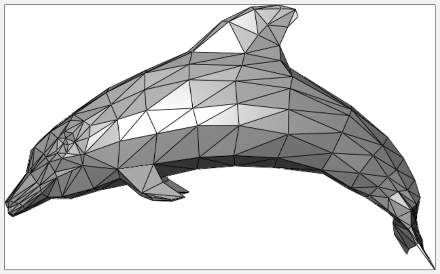
Geometry

Polygonal Meshes

- OpenGL only directly supports points, lines, and triangles
  - any other surface is represented by a *polygonal mesh*



https://commons.wikimedia.org/wiki/File:Dolphin_triangle_mesh.png

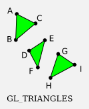
Polygonal Meshes

⚠ syntax is OpenGL 1.0, not WebGL – for WebGL, use one `gl.drawArrays(…)` for each `glBegin`/`glEnd` pair

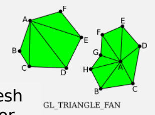
- drawing a polygonal mesh means drawing each of the polygons/triangles in the mesh

```
glBegin(GL_TRIANGLES);
  glVertex3f(…);
  glVertex3f(…);
  glVertex3f(…);
  …
glEnd();
```

(for a triangle mesh)

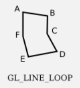


```
glBegin(GL_TRIANGLE_FAN);
  glVertex3f(…);
  glVertex3f(…);
  glVertex3f(…);
  …
glEnd();
glBegin(GL_TRIANGLE_FAN);
  glVertex3f(…);
  glVertex3f(…);
  glVertex3f(…);
  …
glEnd();
…
```



(for a polygonal mesh – one triangle fan for each polygon)

```
glBegin(GL_LINE_LOOP);
  glVertex3f(…);
  glVertex3f(…);
  glVertex3f(…);
  …
glEnd();
glBegin(GL_LINE_LOOP);
  glVertex3f(…);
  glVertex3f(…);
  glVertex3f(…);
  …
glEnd();
…
```

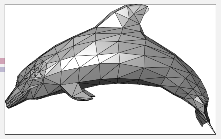


(for a polygonal mesh – one line loop for each polygon)

Efficiency Considerations

- efficiency of representation
  - indexed face sets
- efficiency of execution
  - `glDrawArrays` vs `glDrawElements`

Indexed Face Sets (IFS)



Represent a polygonal mesh with –

- a list of all of the vertices in the mesh
- a list of all of the faces in the mesh
  - each face lists the vertices belonging to that face, identified by the vertex's index in the vertex list
  - vertices are in CCW order when looking at the front of the face

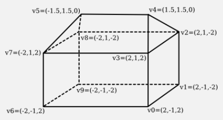
avoids repeated storage of shared vertices

Commonly implemented with arrays –

⚠ syntax is Java, not JavaScript

```
double[][] vertexList =
        {  {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}  };

int[][] faceList =
        {  {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}  };
```



https://commons.wikimedia.org/wiki/File:Dolphin_triangle_mesh.png

glDrawElements

- `glDrawElements` directly supports an indexed face set representation

```
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 of values per vertex, data type)

   // set up buffer - element indices
   gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, index_buffer);
   gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(faces.flat()), gl.STREAM_DRAW);  // copy data from js var to VBO

   // draw the primitives
   for (let i = 0, face = 0; face < faces.length; i += faces[face].length, face++) {
       gl.drawElements(gl.TRIANGLE_FAN, faces[face].length, gl.UNSIGNED_BYTE, i * Uint8Array.BYTES_PER_ELEMENT);
   }
```

indexed face set representation

`.flat()` turns array-of-arrays into a 1D array

need another buffer for the faces

parameters are
– the type of primitive
– the number of elements to render
– the type of values in the element array buffer
– byte offset in the element array buffer

glDrawArrays vs glDrawElements

```
// values for shader attributes
let coords = new Float32Array(
  [
    -0.9, -0.8, 0.0,
    0.9, -0.8, 0.0,
    0, 0.9, 0.0
  ]
);
```

direct representation

```
// 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, coords, 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 of values per vertex, data type)
```

```
// draw the primitives

gl.drawArrays(gl.TRIANGLES, 0, 3);  // first vertex, number of vertices to use
```

```
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 of values per vertex, data type)

   // set up buffer - element indices
   gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, index_buffer);
   gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(faces.flat()), gl.STREAM_DRAW);  // copy data from js var to VBO

   // draw the primitives
   for (let i = 0, face = 0; face < faces.length; i += faces[face].length, face++) {
       gl.drawElements(gl.TRIANGLE_FAN, faces[face].length, gl.UNSIGNED_BYTE, i * Uint8Array.BYTES_PER_ELEMENT);
   }
```

indexed face set representation

glDrawArrays and glDrawElements

```
private final static double[][] dodecVertices =
    { { -0.650000, 0.000000, -0.248278 }, { 0.401722, 0.401722, 0.401722 },
      { 0.650000, 0.000000, 0.248278 }, { 0.401722, -0.401722, 0.401722 },
      { 0.000000, -0.248278, 0.650000 }, { 0.000000, 0.248278, 0.650000 },
      { 0.650000, 0.000000, -0.248278 }, { 0.401722, 0.401722, -0.401722 },
      { 0.248278, 0.650000, 0.000000 }, { -0.248278, 0.650000, 0.000000
      { -0.401722, 0.401722, -0.401722 }, { 0.000000, 0.248278, -0.6500
      { 0.401722, -0.401722, -0.401722 }, { 0.248278, -0.650000, 0.0000
      { -0.248278, -0.650000, 0.000000 }, { -0.650000, 0.000000, 0.2482
      { -0.401722, 0.401722, 0.401722 }, { -0.401722, -0.401722, -0.401
      { 0.000000, -0.248278, -0.650000 }, { -0.401722, -0.401722, 0.401
      { 0.000000, 1.051722, 0.650000 }, { -0.000000, 1.051722, -0.65000
      { 1.051722, 0.650000, -0.000000 }, { 1.051722, -0.650000, -0.0000
      { -0.000000, -1.051722, -0.650000 }, { -0.000000, -1.051722, 0.65
      { 0.650000, 0.000000, 1.051722 }, { -0.650000, 0.000000, 1.051722
      { 0.650000, -0.000000, -1.051722 }, { -0.650000, 0.000000, -1.051
      { -1.051722, 0.650000, -0.000000 },
      { -1.051722, -0.650000, 0.000000 } };

private static int[][] dodecTriangles =
    { { 16, 9, 20 }, { 9, 8, 20 }, { 8, 1, 20 }, {
      { 9, 10, 21 }, { 10, 11, 21 }, { 11, 7, 21 },
      { 8, 9, 21 }, { 8, 7, 22 }, { 7, 6, 22 }, { 6,
      { 1, 8, 22 }, { 6, 12, 23 }, { 12, 13, 23 }, {
      { 3, 2, 23 }, { 2, 6, 23 }, { 18, 17, 24 }, {
      { 14, 13, 24 }, { 13, 12, 24 }, { 12, 18, 24 }, { 1
      { 19, 4, 25 }, { 4, 3, 25 }, { 3, 13, 25 }, { 13, 1
      { 4, 5, 26 }, { 5, 1, 26 }, { 1, 2, 26 }, { 2, 3, 2
      { 15, 16, 27 }, { 16, 5, 27 }, { 5, 4, 27 }, { 4, 1
      { 19, 15, 27 }, { 7, 11, 28 }, { 11, 18, 28 }, { 18
      { 12, 6, 28 }, { 6, 7, 28 }, { 10, 0, 29 }, { 0, 17
      { 17, 18, 29 }, { 18, 11, 29 }, { 11, 10, 29 }, { 0, 10, 30 },
      { 10, 9, 30 }, { 9, 16, 30 }, { 16, 15, 30 }, {
      { 17, 0, 31 }, { 0, 15, 31 }, { 15, 19, 31 }, {
      { 14, 17, 31 } };
```

three alternatives for drawing an IFS triangle mesh

```
let coords_array = [];

for (let face = 0; face < faces.length; face++) {
    for (let i = 0; i < faces[face].length; i++) {
        let vertex = faces[face][i];
        coords_array.push(coords[vertex]);
    }
}
```

⚠ JavaScript

build the direct representation (an array of all of the vertices) and use `drawArrays(gl.TRIANGLES,…)`

```
gl2.glEnableClientState(GL2.GL_VERTEX_ARRAY);
// 3 because (x,y,z) coordinates
gl2.glVertexPointer(3,GL2.GL_DOUBLE,0,
                    Buffers.newDirectDoubleBuffer(flatten(dodecVertices)));
```

setup

```
for ( int i = 0 ; i < dodecTriangles.length ; i++ ) {
  gl2.glDrawElements(GL2.GL_TRIANGLE_FAN,dodecTriangles[i].length,
                     GL2.GL_UNSIGNED_INT,
                     Buffers.newDirectIntBuffer(dodecTriangles[i]));
}
```

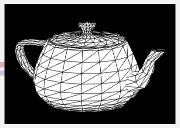
draw each face as a triangle fan

```
gl2.glDrawElements(GL2.GL_TRIANGLES,dodecTriangles.length * 3,
                   GL2.GL_UNSIGNED_INT,
                   Buffers.newDirectIntBuffer(flatten(dodecTriangles)));
```

draw the whole mesh as a list of triangles

⚠ most syntax is Java/OpenGL 1.0, not JavaScript/WebGL

OpenGL Nuts and Bolts



- 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