

# Element Array & External 3D Model & GLSL Syntax

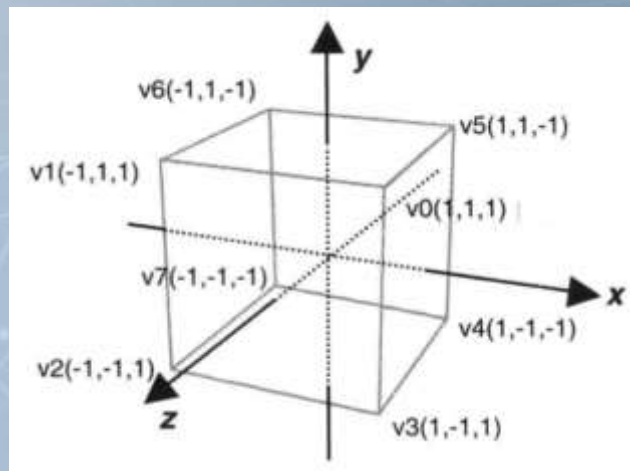
CSU0021: Computer Graphics



Element Array

# Draw a Cube

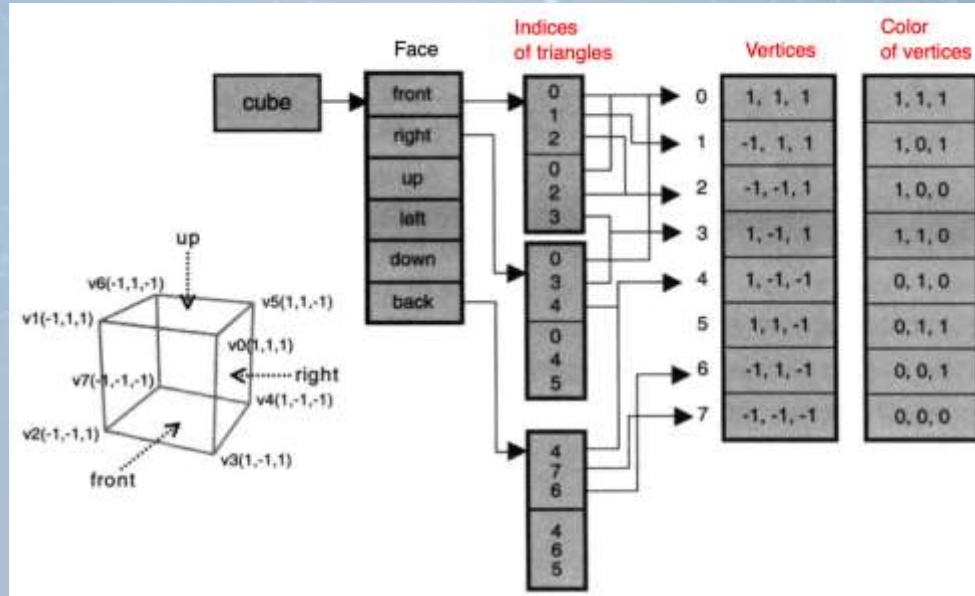
- A cube consists of 6 faces and each face may consist of 2 triangles.
- You have to define and pass  $6 \times 2 \times 3 = 36$  vertices to VBO and call `gl.drawArrays(gl.TRIANGLES, 0, 36)`
- Many vertices are redundantly defined
  - This way to define an object really wastes the storage



```
var vertices = new Float32Array([
    1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, // v0, v1, v2
    1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, // v0, v2, v3
    1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, // v0, v3, v4
    ...
])
```

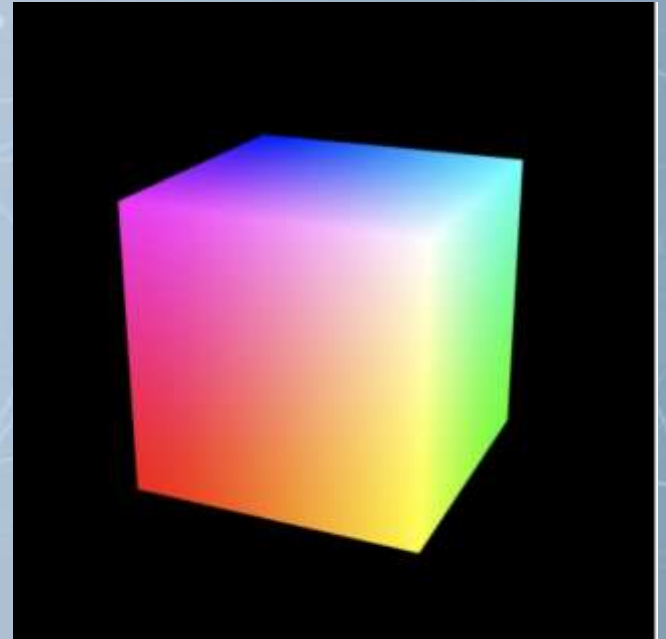
# Element Array

- Define the exact same number of vertices of the object
- Still use triangle to define the connectivity. So, construct triangles by indexing to these vertices



## Example (Ex07-1)

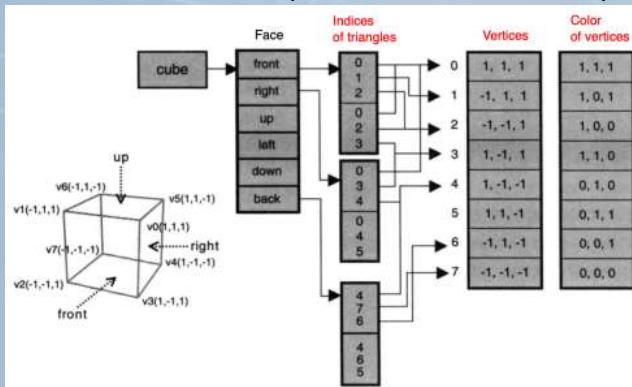
- Draw a colored cube
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js





# Example (Ex07-1)

- `initVertexBuffers()` in WebGL.js
- Prepare the vertex and index arrays
  - We define 8 vertices in “vertices”
  - Each row of “vertices” array includes “x, y, z, r, g, b” of a vertex
  - “indices” array:
    - each row is a face of the cube
    - Each index points to “vertices” array

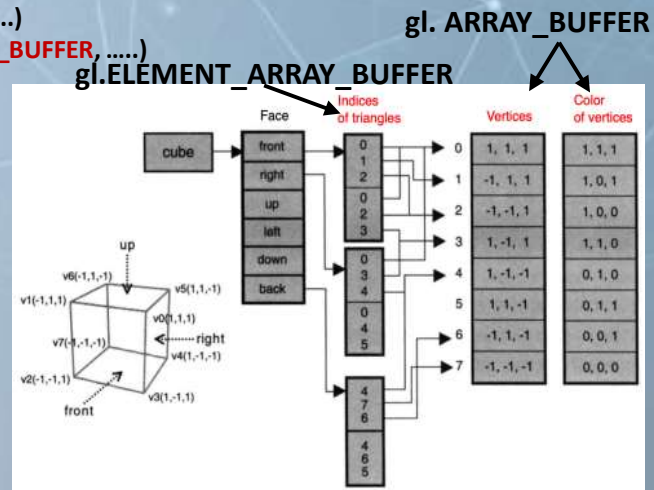


```
function initVertexBuffers(gl, program){
  var vertices = new Float32Array(
    [
      1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
      -1.0, 1.0, 1.0, 1.0, 0.0, 1.0,
      -1.0, -1.0, 1.0, 1.0, 0.0, 0.0,
      1.0, -1.0, 1.0, 1.0, 1.0, 0.0,
      1.0, -1.0, -1.0, 0.0, 1.0, 0.0,
      1.0, 1.0, -1.0, 0.0, 1.0, 1.0,
      -1.0, 1.0, -1.0, 0.0, 0.0, 1.0,
      -1.0, -1.0, -1.0, 0.0, 0.0, 0.0
    ]
  );

  var indices = new Uint8Array(
    [
      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, //bottom
      4, 7, 6, 4, 6, 5 //back
    ]
  );
}
```

# Array Buffer and Element Array Buffer Creation and Rendering

- Steps to create buffers
  - Create array buffers for “vertices” and “color of vertices” (the same as usual)
    - Create a buffer: `gl.createBuffer()`
    - Bind the buffer: `gl.bindBuffer(gl.ARRAY_BUFFER, ....)`
    - Write vertices information to the buffer: `gl.bufferData(gl.ARRAY_BUFFER, ....)`
    - Assign the buffer to an “attribute” variable in vertex shader: `gl.vertexAttribPointer()`
    - Enable the attribute variable: `gl.enableVertexAttributeArray()`
  - Create element buffer for “indices of triangles”
    - Create buffer: `gl.createBuffer()`
    - Bind the buffer: `gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, ....)`
    - Write indices to the buffer: `gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, ....)`
- Rendering
  - Call `gl.drawElements()`
    - WebGL will look for the buffer which bind with element array buffer pointer



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

- <https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/drawElements>
- mode
  - gl.POINTS, gl.LINE\_STRIP, gl.LINE\_LOOP, gl.LINES, gl.TRIANGLE\_STRIP, gl.TRIANGLE\_FAN, gl.TRIANGLES
- count:
  - the number of elements to be rendered
- type:
  - type of values in the element array buffer (usually, gl.UNSIGNED\_BYTE or gl.UNSIGNED\_SHORT)
- offset:
  - a byte offset in the element array buffer (the first index in the element array to draw)



# Example (Ex07-1)

- `initVertexBuffer()` in `WebGL.js`
- Array Buffer and Element Array Buffer Creation

An array buffer to store vertices information and assign to “a\_Position” variable in shader

An array buffer to store color information and assign to “a\_Color” variable in shader

An **element** array buffer to store index information

```
function initVertexBuffers(gl, program){
  var vertices = new Float32Array(
    [
      1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
      -1.0, 1.0, 1.0, 1.0, 0.0, 1.0,
      -1.0, -1.0, 1.0, 1.0, 0.0, 0.0,
      1.0, -1.0, 1.0, 1.0, 1.0, 0.0,
      1.0, -1.0, -1.0, 0.0, 1.0, 0.0,
      1.0, 1.0, -1.0, 0.0, 1.0, 1.0,
      -1.0, 1.0, -1.0, 0.0, 0.0, 1.0,
      -1.0, -1.0, -1.0, 0.0, 0.0, 0.0
    ]
  );
}
```

Note: type we use is unsigned byte

```
var indices = new Uint8Array(
  [
    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, //bottom
    4, 7, 6, 4, 6, 5 //back
  ]
)
```

```
var FSIZE = vertices.BYTES_PER_ELEMENT;
```

```
var vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
var a_Position = gl.getAttribLocation(program, 'a_Position');
gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, FSIZE*6, 0);
gl.enableVertexAttribArray(a_Position);
```

```
var colorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
var a_Color = gl.getAttribLocation(program, 'a_Color');
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*6, FSIZE*3);
gl.enableVertexAttribArray(a_Color);
```

```
var indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
```

```
return indices.length;
```

```
}
```

# Example (Ex07-1)

- main() in WebGL.js
- Ask shader to render

Set up and pass mvp matrix to shader

Set up and pass mvp matrix to shader

```
function main(){
    //Get the canvas context
    var canvas = document.getElementById('webgl');
    var gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    var n = initVertexBuffers(gl, program);

    gl.clearColor(0,0,0,1);
    gl.enable(gl.DEPTH_TEST);

    var u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');

    var mvpMatrix = new Matrix4();//Identity matrix
    mvpMatrix.setPerspective(30, 1, 1, 100); //Projection matrix
    mvpMatrix.lookAt(3, 3, 7, 0, 0, 0, 0, 1, 0); //Projection matrix + View matrix

    gl.uniformMatrix4fv(u_MvpMatrix, false, mvpMatrix.elements);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
}
```

The type we use for the element array is unsigned type

## Let' Try (5mins)

- Check the two variables, “vertices” and “indices”, to make sure you know element array well

The image features a dark blue background with a complex, abstract 3D structure. A prominent wireframe mesh, composed of numerous white lines and dots, forms a jagged, mountain-like silhouette across the upper half of the frame. Below this, a solid, three-dimensional blue shape with sharp, angular edges follows a similar jagged profile. The overall aesthetic is technological and futuristic, suggesting a digital or 3D modeling environment.

External 3D Model

# External Model (Mesh)

- Someone already defines an object for you
  - A mesh is usually approximated by many triangles
  - The model may include
    - Vertices of triangle
    - Normal vector of vertices
    - Texture coordinate of vertices
    - Index array
      - They usually use the way **similar** to “element array” to store the model
- Many formats
  - 3ds, **obj**, json.....



# Cube in Obj File

- Introduction of Obj file format:  
<http://paulbourke.net/dataformats/obj/>
- This cube obj file define
  - Vertices
  - Texture coordinates
  - Normal vector
  - Face indices

Vertex (v)

Texture  
coordinate (vt)

Normal  
vector (vn)

Face (f)

```
# Blender v2.80 (sub 75) OBJ File: ''
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.375000 0.250000
vt 0.375000 0.250000
vt 0.625000 0.250000
vt 0.625000 0.250000
vt 0.375000 0.500000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.625000 1.000000
vt 0.375000 1.000000
vt 0.125000 0.500000
vt 0.375000 0.500000
vt 0.375000 0.750000
vt 0.125000 0.750000
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1/1/1 5/2/1 /3/1 3/4/1
f 4/5/2 3/6/2 7/7/2 8/8/2
f 8/8/3 7/7/3 5/9/3 6/10/3
f 6/10/4 2/11/4 4/12/4 8/13/4
f 2/14/5 1/15/5 3/16/5 4/17/5
f 6/18/6 5/19/6 1/20/6 2/11/6
```

# Cube in Obj File

- “#”: comment
- “v” is vertex position
- “vt” is texture coordinate but we do not use it this week
- “vn” is normal vector
- “f”: faces
  - A cube has 6 face
  - Each face has multiple 4 vertices
- WebGL needs a face defined by a triangle (3 vertices)
  - We can subdivide 1 quad into 2 triangles by ourself

Vertex (v)

Texture  
coordinate (vt)

Normal  
vector (vn)

Face (f)

```
# Blender v2.80 (sub 75) OBJ File: ''
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.625000 0.250000
vt 0.375000 0.250000
vt 0.375000 0.250000
vt 0.625000 0.250000
vt 0.625000 0.500000
vt 0.375000 0.500000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.625000 1.000000
vt 0.375000 1.000000
vt 0.125000 0.500000
vt 0.375000 0.500000
vt 0.375000 0.750000
vt 0.125000 0.750000
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1/1/1 5/2/1 7/3/1 1/4/1
f 4/5/2 3/6/2 7/7/2 8/8/2
f 8/8/3 7/7/3 5/9/3 6/10/3
f 6/10/4 2/11/4 4/12/4 8/13/4
f 2/14/5 1/15/5 3/16/5 4/17/5
f 6/18/6 5/19/6 1/20/6 2/11/6
```

4 vertices per face

# Cube in Obj File

- A vertex of a face is defined by indices which points to “v”, “vt”, “vn” ...
- A vertex could have multiple properties
  - f 1 5 7 3 (vertex)
  - f 1/1 5/2 7/3 3/4 (vertex/texCoordi)
  - f 1/1/1 5/2/1 7/3/1 3/4/1 (vertex/texCoordi/normal)**
  - f 1//1 5//1 7//1 3//1 (vertex/normal)

Vertex (v)

Texture coordinate (vt)

Normal vector (vn)

Face (f)

```
# Blender v2.80 (sub 75) OBJ File: ''
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.625000 0.250000
vt 0.375000 0.250000
vt 0.375000 0.250000
vt 0.625000 0.250000
vt 0.625000 0.500000
vt 0.375000 0.500000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.625000 1.000000
vt 0.375000 1.000000
vt 0.125000 0.500000
vt 0.375000 0.500000
vt 0.375000 0.750000
vt 0.125000 0.750000
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1/1/1 5/2/1 7/3/1 3/4/1
f 4/5/2 3/6/2 1/7/2 8/8/2
f 8/8/3 7/7/3 5/9/3 6/10/3
f 6/10/4 2/11/4 4/12/4 8/13/4
f 2/14/5 1/15/5 3/16/5 4/17/5
f 6/18/6 5/19/6 1/20/6 2/11/6
```

# Cube in Obj File

- The vertex 5/2/1 consists of
  - vertex with index 5
  - texture coordinate with index 2
  - normal vector with index 1
- No color?
  - Yes, no color
  - In most of 3D models, we use an alternative way (texture coordinate and texture image) to define color. That is the topic for next week.

Vertex (v)

Texture  
coordinate (vt)

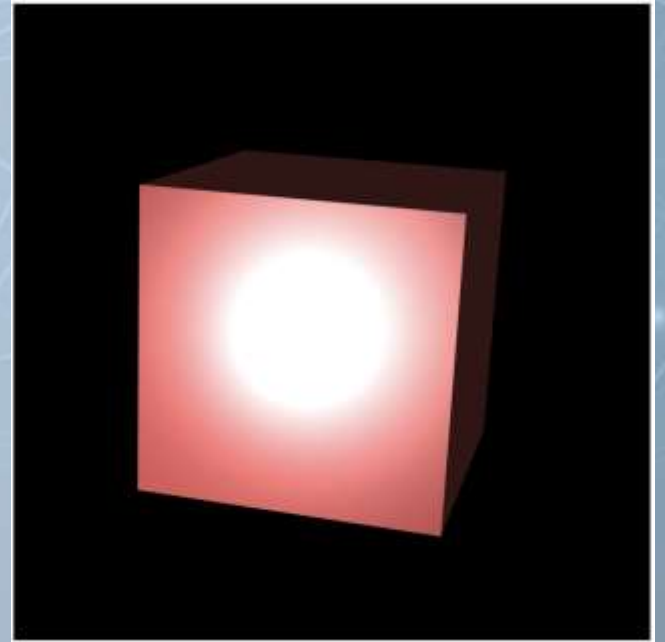
Normal  
vector (vn)

Face (f)

```
# Blender v2.80 (sub 75) OBJ File: ''
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.375000 0.250000
vt 0.375000 0.250000
vt 0.625000 0.250000
vt 0.625000 0.500000
vt 0.375000 0.500000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.625000 1.000000
vt 0.375000 1.000000
vt 0.125000 0.500000
vt 0.375000 0.500000
vt 0.375000 0.750000
vt 0.125000 0.750000
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1/1/1 5/2/1 7/3/1 3/4/1
f 4/5/2 3/6/2 7/7/2 8/8/2
f 8/8/3 7/7/3 5/9/3 6/10/3
f 6/10/4 2/11/4 4/12/4 8/13/4
f 2/14/5 1/15/5 3/16/5 4/17/5
f 6/18/6 5/19/6 1/20/6 2/11/6
```

## Example (Ex07-2)

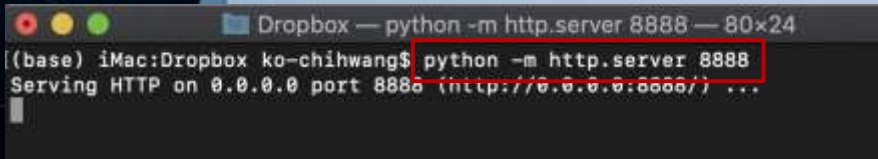
- Load the cube obj file and render it
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js
  - cube.obj



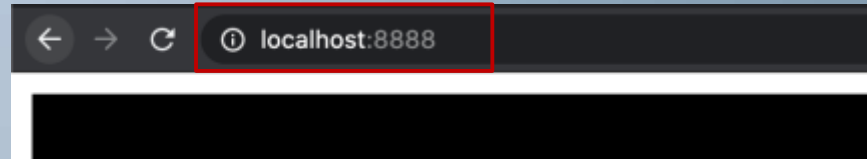


## Example (Ex07-2)

- In this example, the browser will load the file, “cube.obj”
- Because of security issue, your browser will block it (file loading) if you directly drag index.html into your browser
- There are multiple ways to solve this issue
  - One of them is using Python to create a local http server and let browser connect to localhost
    - Open terminal (depend on your OS, you may have different way to open terminal and run python)
    - Run “python -m http.server 8888”. This command creates a simple http server at port 8888
    - Open browser, link to “localhost:8888”



```
Dropbox — python -m http.server 8888 — 80x24
(base) iMac:Dropbox ko-chihwang$ python -m http.server 8888
Serving HTTP on 0.0.0.0 port 8888 (http://0.0.0.0:8888/) ...
```



# Example (Ex07-2)

- parseOBJ() in WebGL.js
  - This function can parse obj files for you
- You pass text of an obj file into this function.
- This function will parse it for you and return the result.
  - Convert quads into triangles
    - Store data by the basic **vertex array** instead of the element array
  - Separate vertex positions, texture coordinates and normal vectors in different javascript arrays
- This function comes from <https://webglfundamentals.org/webgl/lessons/webgl-load-obj.html>
  - You can find more explanations there

```
function parseOBJ(text) {  
  // because we don't know how many lines we'll have I let's just fill in the max data  
  const objPositions = [0, 0, 0];  
  const objTexcoords = [0, 0];  
  const objNormals = [0, 0, 0];  
  
  // Same order as 'F' indices  
  const objVertexData = []  
  objPositions,  
  objTexcoords,  
  objNormals,  
];  
  
  // same order as 'F' indices  
  let webglVertexData = []  
  [], // positions  
  [], // texcoords  
  [], // normals  
];  
  
  function newGeometry() {  
    // If there is an existing geometry and it's  
    // not empty then start a new one.  
    if (geometry && geometry.data.position.length) {  
      geometry = undefined;  
    }  
    setGeometry();  
  }  
  
  function addVertex(vertex) {
```

```
    if (!v) {  
      continue;  
    }  
  
    const [, keyword, unparsedArgs] = v;  
    const parts = line.split(/\s+/).slice(1);  
    const handler = keywords[keyword];  
    if (!handler) {  
      console.warn('unhandled keyword:', keyword); // eslint-disable-line  
      continue;  
    }  
    handler(parts, unparsedArgs);  
  }  
  
  return {  
    position: webglVertexData[0],  
    texcoord: webglVertexData[1],  
    normal: webglVertexData[2],  
  };  
}
```

You have to put await execution in an async function

## Example (Ex07-2)

- main() in WebGL.js
- Load obj file and get text

Load the file: you need await keyword to wait the loading finish and continue

Pass the obj text to the parseOBJ()

Use data from Obj file to initialize vertex buffers

```
async function main() {
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    program.u_normalMatrix = gl.getUniformLocation(program, 'u_normalMatrix');
    program.u_lightPosition = gl.getUniformLocation(program, 'u_lightPosition');
    program.u_ViewPosition = gl.getUniformLocation(program, 'u_ViewPosition');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');
    program.u_Kd = gl.getUniformLocation(program, 'u_Kd');
    program.u_Ks = gl.getUniformLocation(program, 'u_Ks');
    program.u_shininess = gl.getUniformLocation(program, 'u_shininess');
    program.u_Color = gl.getUniformLocation(program, 'u_Color');

    response = await fetch('cube.obj');
    text = await response.text();
    obj = parseOBJ(text);
    nVertex = initVertexBuffers(gl, program,
        new Float32Array( obj.position ),
        new Float32Array( obj.normal ));

    mvpMatrix = new Matrix4();
    modelMatrix = new Matrix4();
    normalMatrix = new Matrix4();

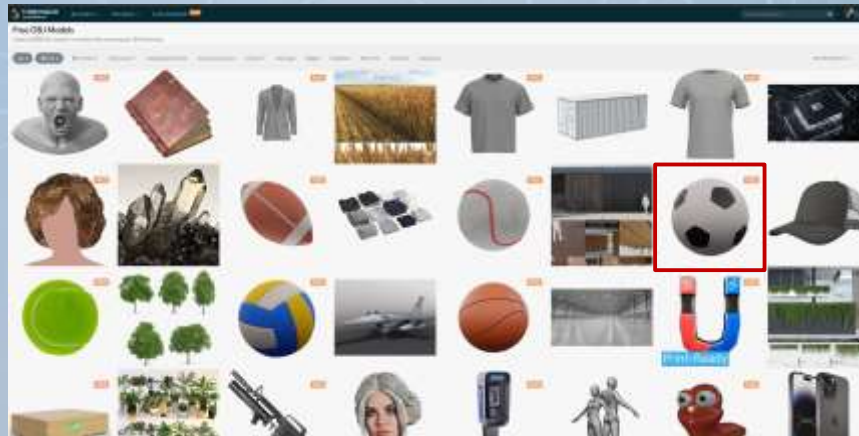
    gl.enable(gl.DEPTH_TEST);

    draw();//draw it once before mouse move

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

# Let's Try (10mins)

- Make sure you can run this example and see the result
- Load and render another 3D Model: mario.obj
- More 3D models in <https://www.turbosquid.com/>
  - You may have to register a <https://www.turbosquid.com/> account
  - Click “OBJ MODEL” icon. Click “Price”, select “free”. Find “Football Ball” (soccer)
  - Click “download” -> click “show all” -> Click “Football ball.obj” and download
- **Ex07-2 may not work for every models**

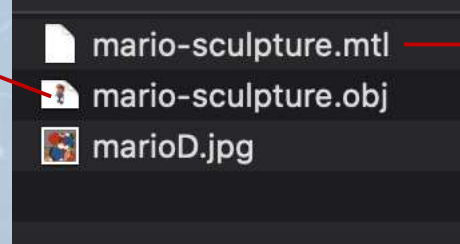




## More for Obj Format

- If you download .obj from clara.io, you usually have one .obj, one .mtl and some images in the folder
  - .obj: vertex information
  - .mtl: material information (for illumination)
  - images: texture images (color information)

```
1 # Blender v2.74 (sub 0) OBJ File: ''
2 # www.blender.org
3 mtllib mario-sculpture.mtl
4 o mario
5 v 3.717522 39.296425 15.052347
6 v 3.443130 39.293114 15.072504
7 v 3.436722 39.204453 14.664225
8 v 3.720497 39.213699 14.656729
9 v 3.998795 39.225311 14.651037
10 v 3.986992 39.300591 15.034155
11 v 3.978912 39.318314 15.383265
12 v 3.717796 39.316139 15.414076
13 v 3.451859 39.314941 15.447264
14 v 2.684723 39.333046 15.978128
15 v 2.650452 39.314827 15.560564
16 v 2.919006 39.315331 15.516424
```

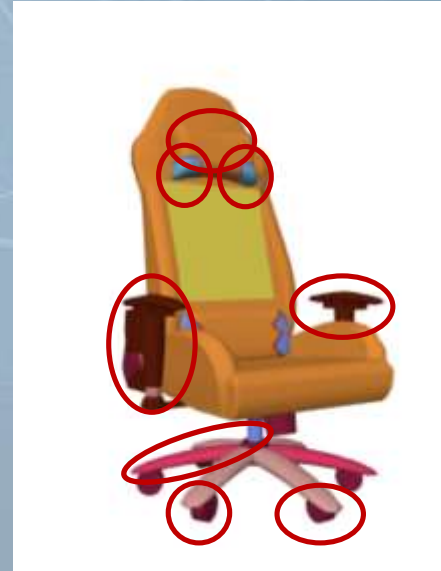


```
1 # Blender MTL File: 'None'
2 # Material Count: 1
3
4 newmtl mario_1
5 Ns 96.078431
6 Ka 0.000000 0.000000 0.000000
7 Kd 1.000000 1.000000 1.000000
8 Ks 0.040000 0.040000 0.040000
9 Ni 1.000000
10 d 1.000000
11 illum 2
12 map_Kd marioD.jpg
13 |
```



# Object with Multiple Pieces (Obj)

- The cube and the mario are special cases
- Many objects consist of multiple pieces
  - The chair: handles, wheels....
- How the obj file describe this type of objects



# Object with Multiple Pieces (Obj)

- Cube.obj contains only one piece

How the multiple pieces obj file looks like

Mtllib \*\*\*\*.mtl

o nameOfPiece1

v ...

v ...

.....

vt ...

vt ...

...

vn ...

vn ...

...

Usemtl Material

s ...

f ...

f ...

.....

o nameOfPiece2

v ...

v ...

.....

vt ...

vt ...

...

vn ...

vn ...

...

Usemtl Material

s ...

f ...

f ...

.....

o nameOfPiece3

.....

Information of the 1<sup>st</sup> piece

Information of the 2<sup>nd</sup> piece

Information of the 3<sup>rd</sup> 4<sup>th</sup> 5<sup>th</sup> .....pieces

usually one obj file one mtlfile to indicate what the material file is

Name of this piece

All information of one piece

How the normal is calculated and whether use the material file (we can ignore them this course)

```
# Blender v2.80 (sub 75) OBJ File: ''
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.625000 0.250000
vt 0.375000 0.250000
vt 0.375000 0.250000
vt 0.625000 0.250000
vt 0.375000 0.500000
vt 0.625000 0.500000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.625000 1.000000
vt 0.375000 1.000000
vt 0.125000 0.500000
vt 0.375000 0.500000
vt 0.375000 0.750000
vt 0.125000 0.750000
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1/1/1 5/2/1 7/3/1 3/4/1
f 4/5/2 3/6/2 7/7/2 8/8/2
f 8/8/3 7/7/3 5/9/3 6/10/3
f 6/10/4 2/11/4 4/12/4 8/13/4
f 2/14/5 1/15/5 3/16/5 4/17/5
f 6/18/6 5/19/6 1/20/6 2/11/6
```

## Example (Ex07-3)

- Load “sonic.obj” and render
  - sonic consists of 13 pieces
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js
  - sonic.obj



# Example (Ex07-3)

- parseOBJ() in WebGL.js
- Pass whole text of the obj file to parseObj(),  
parseObj() will parse multiple pieces object for you
  - More details:  
<https://webglfundamentals.org/webgl/lesson/s/webgl-load-obj.html>

```
response = await fetch('sonic.obj');
text = await response.text();
obj = parseOBJ(text);
console.log(obj);
```

```
{ geometries: Array(13), materialLibs: Array(1) }
  geometries: Array(13)
    0: {
      data: { position: Array(2331), texcoord: Array(1554), normal: Array(2331) }
      groups: ["default"]
      material: "d1419efe_dds"
      object: "DrawCall_0278"
      __proto__: Object
    }
    1: {object: "DrawCall_0270", groups: Array(1), material: "64124be4_dds", data: {...}}
    2: {object: "DrawCall_0266", groups: Array(1), material: "64124be4_dds", data: {...}}
    3: {object: "DrawCall_0276", groups: Array(1), material: "f1f6d3cb_dds", data: {...}}
    4: {object: "DrawCall_0275", groups: Array(1), material: "bab97353_dds", data: {...}}
    5: {object: "DrawCall_0267", groups: Array(1), material: "64124be4_dds", data: {...}}
    6: {object: "DrawCall_0271", groups: Array(1), material: "64124be4_dds", data: {...}}
    7: {object: "DrawCall_0269", groups: Array(1), material: "64124be4_dds", data: {...}}
    8: {object: "DrawCall_0272", groups: Array(1), material: "f1f6d3cb_dds", data: {...}}
    9: {object: "DrawCall_0268", groups: Array(1), material: "64124be4_dds", data: {...}}
    10: {object: "DrawCall_0274", groups: Array(1), material: "bab97353_dds", data: {...}}
    11: {object: "DrawCall_0273", groups: Array(1), material: "f1f6d3cb_dds", data: {...}}
    12: {object: "DrawCall_0277", groups: Array(1), material: "d1419efe_dds", data: {...}}
    length: 13
    __proto__: Array(0)
  materialLibs: ["sonic-the-hedgehog.atl"]
  __proto__: Object
```

```
function parseOBJ(text) {
  // because indices are base 1 let's just fill in the 0th data
  const objPositions = [[0, 0, 0]];
  const objTexcoords = [[0, 0]];
  const objNormals = [[0, 0, 0]];

  // same order as 'f' indices
  const objVertexData = [
    objPositions,
    objTexcoords,
    objNormals,
  ];

  // same order as 'f' indices
  let webglVertexData = [
    [], // positions
    [], // texcoords
    [], // normals
  ];

  const materialLibs = [];
  const geometries = [];
  let geometry;
  let groups = ['default'];
  let material = 'default';
  let object = 'default';

  const noop = () => {};

  function newGeometry() {
    // If there is an existing geometry and it's
    // not empty then start a new one.
    if (geometry && geometry.data.position.length) {
      geometry = undefined;
    }
  }

  function setGeometry() {
    if (!geometry) {
      const position = [];
      const texcoord = [];
    }
  }
}
```



Store all vertex buffer objects of all pieces

## Example (Ex07-3)

- main() in WebGL.js
- Idea of creating VBOs and render image
  - We create a vertex buffer for each piece of the sonic
  - When drawing, we use a for loop to send pieces (bind VBO) to draw one by one

Create vertex buffers for each piece and store them in "objComponents"

```
var cameraX = 3, cameraY = 3, cameraZ = 7;  
var objScale = 0.05;  
var objComponents = [];  
  
async function main(){  
  canvas = document.getElementById('webgl');  
  gl = canvas.getContext('webgl2');  
  if(!gl){  
    console.log('Failed to get the rendering context for WebGL');  
    return;  
  }  
  
  program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);  
  
  gl.useProgram(program);  
  
  program.a_Position = gl.getAttribLocation(program, 'a_Position');  
  program.a_Normal = gl.getAttribLocation(program, 'a_Normal');  
  program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');  
  program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');  
  program.u_normalMatrix = gl.getUniformLocation(program, 'u_normalMatrix');  
  program.u_LightPosition = gl.getUniformLocation(program, 'u_LightPosition');  
  program.u_ViewPosition = gl.getUniformLocation(program, 'u_ViewPosition');  
  program.u_Ka = gl.getUniformLocation(program, 'u_Ka');  
  program.u_Kd = gl.getUniformLocation(program, 'u_Kd');  
  program.u_Ks = gl.getUniformLocation(program, 'u_Ks');  
  program.u_shininess = gl.getUniformLocation(program, 'u_shininess');  
  program.u_Color = gl.getUniformLocation(program, 'u_Color');  
  
  response = await fetch('sonic.obj');  
  text = await response.text();  
  obj = parseOBJ(text);  
  
  for( let i=0; i < obj.geometries.length; i ++ ){  
    let o = initVertexBufferForLaterUse(gl,  
                                         obj.geometries[i].data.position,  
                                         obj.geometries[i].data.normal,  
                                         obj.geometries[i].data.texcoord);  
    objComponents.push(o);  
  }  
  
  mvpMatrix = new Matrix4();  
  modelMatrix = new Matrix4();  
  normalMatrix = new Matrix4();  
  
  gl.enable(gl.DEPTH_TEST);
```

Load obj file and parse it



# Example (Ex07-3)

- draw() in WebGL.js
- Idea of creating VBOs and render image
  - We create a vertex buffer for each piece of the sonic
  - **When drawing, we use a for loop to send pieces (bind VBO) to draw one by one**

```
function draw(){
    gl.clearColor(0,0,0,1);

    //model Matrix (part of the mvp matrix)
    modelMatrix.setRotate(angleY, 1, 0, 0); //for mouse rotation
    modelMatrix.rotate(angleX, 0, 1, 0); //for mouse rotation
    modelMatrix.scale(objScale, objScale, objScale);
    // modelMatrix.translate(0.0, 0.0, -1.0);
    // modelMatrix.scale(1.0, 0.5, 2.0);
    //mvp: projection * view * model matrix
    mvpMatrix.setPerspective(30, 1, 1, 100);
    mvpMatrix.lookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 1, 0);
    mvpMatrix.multiply(modelMatrix);

    //normal matrix
    normalMatrix.setInverseOf(modelMatrix);
    normalMatrix.transpose();

    gl.uniform3f(program.u_LightPosition, 0, 0, 3);
    gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
    gl.uniform1f(program.u_Ka, 0.2);
    gl.uniform1f(program.u_Kd, 0.7);
    gl.uniform1f(program.u_Ks, 1.0);
    gl.uniform1f(program.u_shininess, 10.0);
    gl.uniform3f(program.u_Color, 1.0, 0.4, 0.4);

    gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
    gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    for( let i=0; i < objComponents.length; i ++ ){
        initAttributeVariable(gl, program.a_Position, objComponents[i].vertexBuffer);
        initAttributeVariable(gl, program.a_Normal, objComponents[i].normalBuffer);
        gl.drawArrays(gl.TRIANGLES, 0, objComponents[i].numVertices);
    }
}
```

# Example (Ex07-3)

- Note
  - The obj parser, `parseObj()`, we provide is not a complete obj parser. It may not be able to parse some obj files correctly.
  - After you successfully load an obj file and render it, the object may be too small or large to see it on canvas. If you cannot see the object, try to the scale issue.
  - The origin of the object space may not be at center of the object (ex: at the left bottom corner of the object).

## Let's try (5 mins)

- Browse the code and sonic.obj file to make sure you know what's going on there.
- Go to clara.io and download another multiple pieces obj file to replace sonic.obj



# GLSL (Shader Language) Syntax

# Basic

- Very similar to C language
- Case sensitive
- Strong type language
- One shader has one and only one main()
  - main() has not input arguments and no return value
- Comment: // or /\* ... \*/



# Data Type and Basic Variables

- Variable names
  - Same as C
  - Cannot start with “gl\_”, “webgl\_” or “\_webgl\_”
  - Other keywords

attribute	bool	break	bvec2	bvec3	bvec4
const	continue	discard	do	else	false
float	for	highp	if	in	inout
int	invariant	ivec2	ivec3	ivec4	lowp
mat2	mat3	mat4	medium	out	precision
return	sampler2D	samplerCube	struct	true	uniform
varying	vec2	vec3	vec4	void	while

asm	cast	class	default
double	dvec2	dvec3	dvec4
enum	extern	external	fixed
flat	fvec2	fvec3	fvec4
goto	half	hvec2	hvec3
hvec4	inline	input	interface
long	namespace	noinline	output
packed	public	sampler1D	sampler1DShadow
sampler2DRect	sampler2DRectShadow	sampler2DShadow	sampler3D
sampler3DRect	short	sizeof	static
superp	switch	template	this
typedef	union	unsigned	using
volatile			

# Data Type and Basic Variables

- Basic types
  - bool: “true” or “false”
  - int
  - float
    - no “double”, use “precision” to define the precision of a floating point
    - highp, medium, lowp (we learn this at very beginning of this course)
- Some examples
  - int i = 8;
  - float f1 = 8.0;
  - ~~float f2 = 8;~~
  - ~~float f3 = 8.0f;~~
  - float f4 = float(i);
  - float f5 = float(8);
- Type casting
  - int(float): discard decimal part
  - int(bool): true -> 1, false -> 0
  - float(int)
  - float(bool): true -> 1.0, false -> 0.0
  - bool(int): 0 -> false, other values -> true
  - bool(float): 0.0 -> false, other values -> true

## Basic operators

```
-  
*  
/  
+  
-  
++  
--  
=  
+= -= *= /=  
< > <= >=  
== !=  
!  
&&  
||  
^^  
  
condition?  
expression1:expression2
```

# Matrix and Vector

- Vector type:
  - `vec2`, `vec3`, `vec4` : floating point vector
  - `ivec2`, `ivec3`, `ivec4` : integer vector
  - `bvec2`, `bvec3`, `bvec4` : boolean vector
- Matrix type (floating points only)
  - `mat2` : 2x2 matrix
  - `mat3` : 3x3 matrix
  - `mat4` : 4x4 matrix

# Vector

- Examples of constructors of vector type
  - `vec4 position = vec4(1.0, 2.0, 3.0, 4.0);`
  - `vec3 v3 = vec3(1.0, 0.0, 0.5);`
  - `vec2 v2 = vec2(v3); //v2=(1.0, 0.0)`
  - `vec4 v4 = vec4(1.0); //v4=(1.0,1.0,1.0,1.0)`
  - ~~`vec4 vv = 1.0;`~~
  - `Vec4 v4b = vec4(v2, v4); //v4b=(1.0, 0.0, 1.0,1.0)`

# Vector

- Access elements in vector
- Three syntax sets: just increase the readability of your code
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q
- Examples (basic)
  - `vec3 v3 = vec3(1.0, 2.0, 3.0);`
  - `float f;`
  - `f = v3.x;` (f is 1.0)
  - `f = v3.y;` (f is 2.0)
  - `f = v3.z;` (f is 3.0)
  - `f = v3.r;` (f is 1.0)
  - `f = v3.s;` (f is 1.0)
  - ~~`f = v3.w;` (f is 1.0)~~, v3 has no the 4<sup>th</sup> element
- Examples (swizzling)
  - `vec2 v2;`
  - `v2 = v3.xy;` (v2 is [1.0, 2.0])
  - `v2 = v3.yz;` (v2 is [2.0, 3.0])
  - `v2 = v3.xz;` (v2 is [1.0, 3.0])
  - `v2 = v3.yx;` (v2 is [2.0, 1.0])
  - `v2 = v3.xx;` (v2 is [1.0, 1.0])
  - `vec3 v3a = v3.zyx;` (v3a is [3.0, 2.0, 1.0])
- Assign (swizzling)
  - `vec4 position = vec4(1.0, 2.0, 3.0, 4.0);`
  - `position.xw = vec2(5.0, 6.0);` (position is [5.0, 2.0, 3.0, 6.0])
  - ~~`vec3 a = v3.was;`~~, cannot mix alphabets from different sets
- You can also use [ ] to access any single element in a vector you want (same as accessing 1D array in C)
  - `f = v3[1];` (f is 2.0)



# Matrix

- Column-major

- `mat4 m4 = mat4(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0);`

- $$\bullet \begin{bmatrix} 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \\ 4.0 & 8.0 & 12.0 & 16.0 \end{bmatrix}$$

# Matrix

- Examples of constructors of matrix type

- `vec2 v21 = vec2(1.0, 3.0);`

- `vec2 v22 = vec2(2.0, 4.0);`

- `mat2 m21 = mat2(v21, v22);`  $\begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{bmatrix}$

- `vec4 v4 = vec4(1.0, 3.0, 2.0, 4.0);`

- `mat2 m22 = mat2(v4);`  $\begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{bmatrix}$

- `mat2 m2 = mat2(1.0, 3.0, v22);`

- `mat4 m4 = mat4(1.0);`

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

- ~~`mat4 m44 = mat4(1.0, 2.0, 3.0);`~~

# Matrix

- Access elements in matrix
- `mat4 m4 = mat4(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0);`
- Use bracket, `[ ]`, to access elements in matrix
- `vec4 v4 = m4[0];` (m4 is [1.0, 2.0, 3.0, 4.0])
- `float m23 = m4[1][2];` (m23 is 7.0)
- `Float m32 = m4[2].y;` (m32 is 10.0)
- The index in `[]` has to be **constant index**
  - `vec4 v4 = m4[1];`
  - `const int indexC = 0;`
  - `vec4 v4a = m4[indexC];`
  - `vec4 v4b = m4[indexC+1];`
  - `int index = 0;`
  - ~~`vec4 v4c = m4[index];`~~ (incorrect: index is not const)

$$m4 = \begin{bmatrix} 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \\ 4.0 & 8.0 & 12.0 & 16.0 \end{bmatrix}$$

# Matrix and Vector Operators

- Example of operators
  - `vec3 v3a, v3b, v3c;`
  - `mat3 m3a, m3b, m3c;`
  - `float f;`
- `v3b = v3a + f;`
  - `v3b.x = v3a.x + f`
  - `v3b.y = v3a.y + f`
  - `v3b.z = v3a.z + f`
- `v3c = v3a + v3b;`
  - `v3c.x = v3a.x + v3b.x`
  - `v3c.y = v3a.y + v3b.y`
  - `v3c.z = v3a.z + v3b.z`
- `m3b = m3a + f;`
  - Add `f` to all elements of `m3a` and assign to `m3b`

\*

/

+

-

++

--

=

+=, -=

\*=, /=

==, !=

# Matrix and Vector Operators

- Example of operators
  - `vec3 v3a, v3b, v3c;`
  - `mat3 m3a, m3b, m3c;`
  - `float f;`
- Matrix-vector multiplication
  - `m*v`
    - `v3b = m3a * v3a;`
  - `v*m`
    - `v3b = v3a * m3a;`
- Matrix-matrix multiplication (`m*m`)
  - `m3c = m3a * m3b;`

\*  
/  
+  
-  
++  
--  
=  
+=, -=  
\*=, /=  
==, !=



# Struct

- Examples

```
struct Light{  
    vec4 color;  
    vec3 position;  
}  
Light l1, l2;
```

```
struct Light{  
    vec4 color;  
    vec3 position;  
} l1;
```

Construction:

```
l1 = Light(vec4(1.0,0.0,0.0,1.0), vec3(2.0,6.0,1.2));
```

Access members:

```
vec4 color = l1.color;  
vec3 pos = l1.position;
```

# Array

- Determine the size of arrays at compile time only (cannot at run time)
- Examples
  - `float floatArray[4];`
  - `vec4 vec4Array[2];`
  - `int size = 4;`
  - ~~`vec4 vec4Array[size];`~~ //this determine the size at run time
  - `float f = floatArray[2];`
  - `vec4Array[0] = vec4(4.0,3.0,2.0,1.0);` //also for initialization

# Branch and Loop

- if, else if, else: same as C
- “switch-case”: same as C (but old GLSL version does not support switch-case statement)
- Loop: for, while, do-while
  - break, continue
- “discard”
  - fragment shader only
  - It means that ignore current fragment

# Self-defined Function

- Almost the same as C
- You have to declare the function if you use it before you define it.
- Example:

```
float luma(vec4 color){  
    float r = color.r;  
    float g = color.g;  
    float b = color.b;  
    return 0.21*r + 0.71*g +0.072*b;  
}  
  
attribute vec4 a_Color;  
void main(){  
    ...  
    float brightness = luma(a_Color);  
    ...  
}
```

# Self-defined Function

- Arguments: always pass by value? No
  - in: pass by value
  - out: pass by reference (no value pass in, but value pass out)
  - inout: pass by reference (value pass in and value pass out)
  - No in, out or inout? "in" is the default mode
- Example:

```
vec3 example(in float x, in bool beta, inout int gamma, out int theta)
{
    // statement(s)
}
```

```
vec3 phi = example(3.5, true, delta, chi);
```

  - The first two arguments are pass by value
  - Value in **delta** will be passed into the function and any changes in the **delta**'s value changes the **delta** from the calling statement.
  - the **chi**'s value is initialized by the calling statement and any changes made by the function change the **chi** from the calling statement.



# List of Some Build-in Functions

- radians(), degrees()
- sin(), cos(), tan(), acos(), atan() ...
- pow(), exp(), log(), exp2(), log2(), sqrt(), inversesqrt()
- abs(), min(), max(), mod(), sign(), floor(), ceil(), clamp(), mix(), fract()
- length(), distance(), dot(), cross(), normalize(), reflect(), refract()
- More.....

# Scope and More about Variables

- Similar to C, GLSL has the concept of variable scope
- `const`: const variable
- `attribute`
- `uniform`
- `varying`

# Macro and Preprocessor

`#ifdef`

`#if`

`#ifndef`

`#define`

`#undef`

.....

These slides are not complete GLSL syntax tutorial.  
So, you cannot learn GLSL language structurally here.

We just quickly provide some information and  
examples which may be useful for you.  
My purpose is to give you the sense about some  
functions or syntax may exist and give you a chance  
to check the document.