

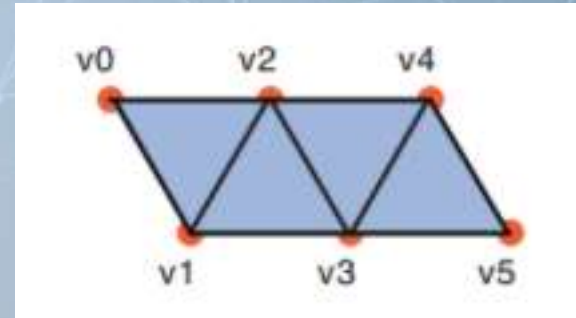
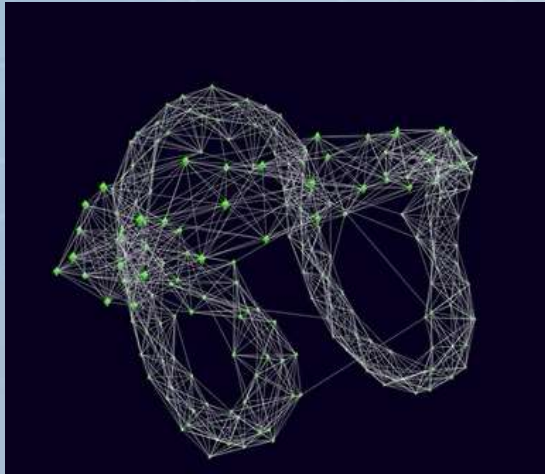


# Vertex Buffer Object (VBO)

CSU0021: Computer Graphics

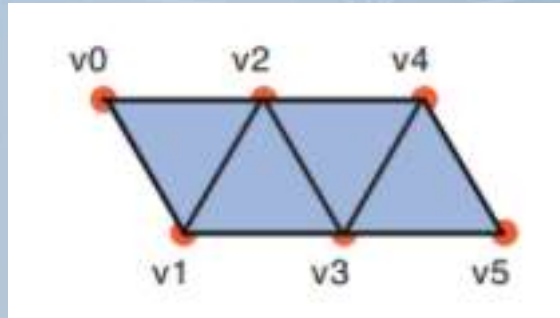
# What if ...

- You want to draw a complicated scene with thousands of vertices
  - Pass them to shader and run one by one using a javascript for loop?
    - This implementation does not utilize the resource of graphics card well
    - Shaders does not know the connection between vertices to render a surface



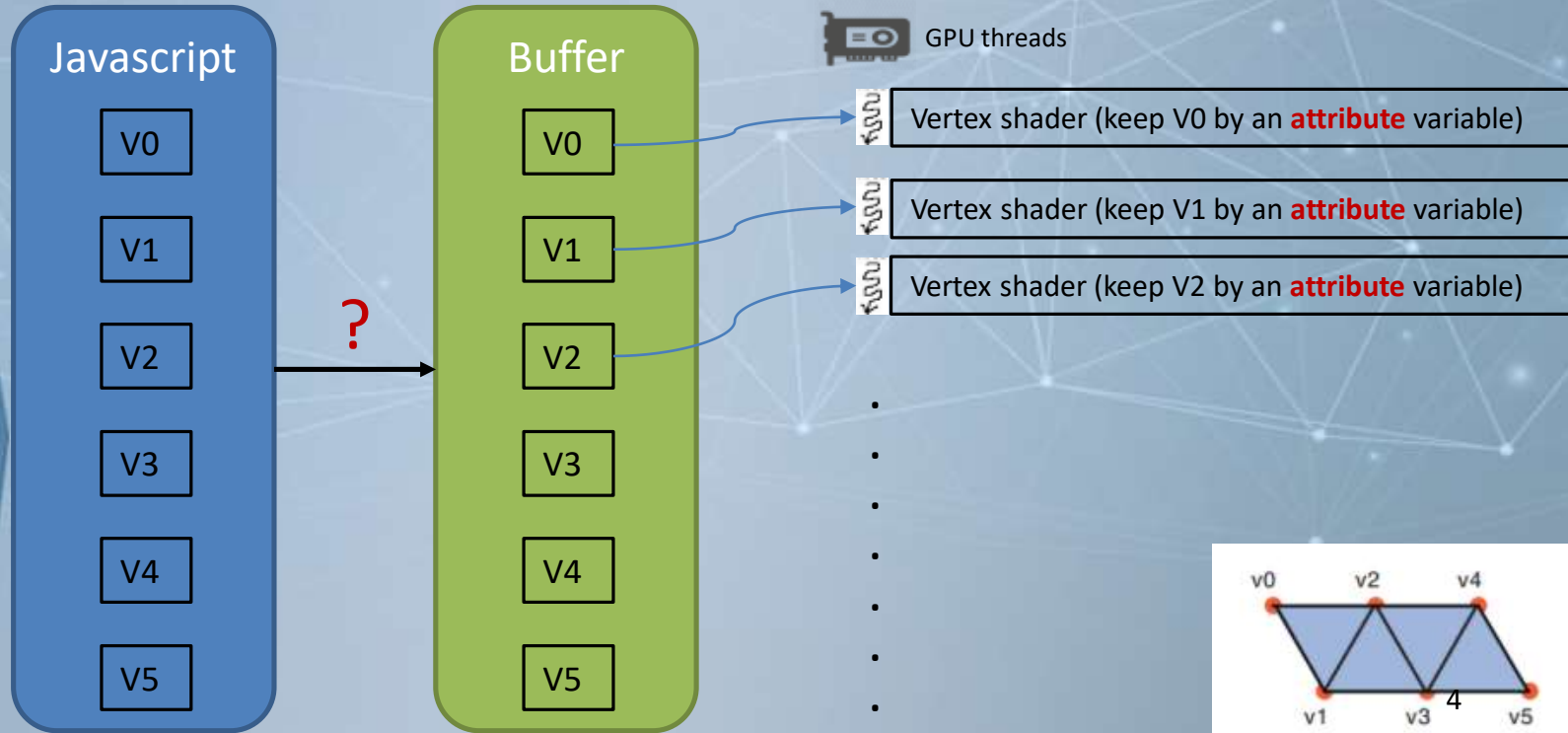
# What We Want ...

- Prepare all vertices (v0 ... v5) in javascript and pass to graphics card (shader)
- Run shaders just once (call "gl.drawArrays()") to render all triangles for us
- "Vertex Buffer Object" (VBO) can help



# Vertex Buffer Object (VBO)

- VBO is a buffer for you to keep the vertices information for drawing



# Vertex Buffer Object (VBO)

- VBO is a buffer for you to keep the vertices information for drawing
- To use VBO, there are multiple steps
  - Create a buffer: **gl.createBuffer()**
  - Bind the buffer: **gl.bindBuffer()**
  - Write vertices information to the buffer: **gl.bufferData()**
  - Assign the buffer to an “attribute” variable in vertex shader: **gl.vertexAttribPointer()**
  - Enable the attribute variable: **gl.enableVertexAttribArray()**
- “attribute” is a keyword for variable in vertex shader
  - Similar to “uniform”
  - Let’s see what the difference between “uniform” and “attribute”

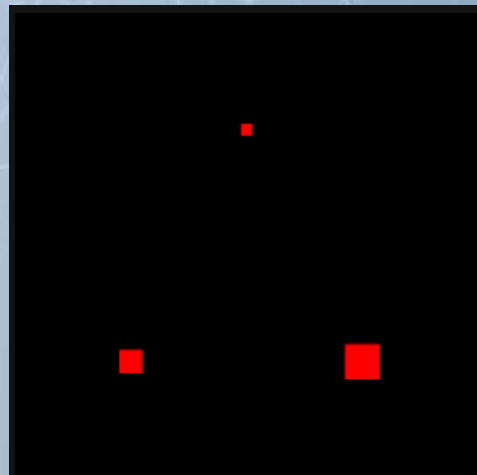
# “uniform” and “attribute” in Shaders

- “uniform”
  - can be used in both **vertex and fragment shader**
  - The same uniform variable of all threads has the **same** value
    - ex: **uniform float var**, and we pass “1.5” to var using `gl.uniform1f()`
    - All “var” in different threads has the same value, 1.5
- “attribute”
  - Only for **vertex shader**
  - The same attribute variable of all threads suppose to have **different** values
    - ex: **attribute vec3 var**
  - So, what the values of “var” in each threads are?



## Example (Ex02-1)

- Draw three points with different sizes
- Call `gl.drawArrays()` just once
- Vertex shader should receive positions and sizes of these points
- Files:
  - `Index.html`
  - `WebGL.js`



# Example (Ex02-1)

- WebGL.js: vertex and fragment shaders
- We have three points (three positions and sizes)
  - E.g. `a_Position` stores the position of one point
  - What happens in the vertex shader?

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute float a_PointSize;
    void main(){
        gl_Position = a_Position;
        gl_PointSize = a_PointSize;
    }
`;

var FSHADER_SOURCE = `
    void main(){
        gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
    }
`;
```



# Example (Ex02-1)

- Shaders runs in **data parallelism** mechanism
  - More details:  
[https://en.wikipedia.org/wiki/Data\\_parallelism](https://en.wikipedia.org/wiki/Data_parallelism)
- In short, each thread runs the same code but different input data
  - In our vertex shader example, each thread runs the same vertex shader code, but the data in their “a\_Position” and “a\_PointSize” are different

# Example (Ex02-1)



GPU threads

Values in `a_Position` and `a_PointSize` of this thread are position and size of point 1

```
var VSHADER_SOURCE = `
attribute vec4 a_Position;
attribute float a_PointSize;
void main(){
    gl_Position = a_Position;
    gl_PointSize = a_PointSize;
};
```

Values in `a_Position` and `a_PointSize` of this thread are position and size of point 2

```
var VSHADER_SOURCE = `
attribute vec4 a_Position;
attribute float a_PointSize;
void main(){
    gl_Position = a_Position;
    gl_PointSize = a_PointSize;
};
```

Values in `a_Position` and `a_PointSize` of this thread are position and size of point 3

```
var VSHADER_SOURCE = `
attribute vec4 a_Position;
attribute float a_PointSize;
void main(){
    gl_Position = a_Position;
    gl_PointSize = a_PointSize;
};
```

**How to make this happen in WebGL???**

# Example (Ex02-1)

- WebGL.js: main()

```
function main(){  
    var canvas = document.getElementById('webgl');  
  
    var gl = canvas.getContext('webgl2');  
    if(!gl){  
        console.log('Failed to get the rendering context for WebGL');  
        return ;  
    }  
  
    let renderProgram = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);  
  
    gl.useProgram(renderProgram);  
  
    var n = initVertexBuffers(gl, renderProgram);  
  
    gl.clearColor(0.0, 0.0, 0.0, 1.0);  
    gl.clear(gl.COLOR_BUFFER_BIT);  
  
    gl.drawArrays(gl.POINTS, 0, n);  
}
```

Self-defined function using  
VBO to pass vertex information

We only call gl.drawArrays()  
once here

# Example (Ex02-1)

- WebGL.js: `initVertexBuffers()`

```
function initVertexBuffers(gl, program){  
  var n = 3;  
  
  var vertices = new Float32Array(  
    [0.0, 0.5, 10.0,    //point0: x, y, size  
    -0.5, -0.5, 20.0,   //point1: x, y, size  
    0.5, -0.5, 30.0]    //point2: x, y, size  
  );  
  var vertexBuffer = gl.createBuffer();  
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);  
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);  
  var FSIZE = vertices.BYTES_PER_ELEMENT;  
  
  var a_Position = gl.getAttribLocation(program, 'a_Position');  
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);  
  gl.enableVertexAttribArray(a_Position);  
  
  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');  
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);  
  gl.enableVertexAttribArray(a_PointSize);  
  
  return n;  
}
```

Number of vertices we will draw

Vertex information in javascript array

# Example (Ex02-1)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

- Check document for gl.createBuffer():
  - <https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/createBuffer>

gl.ARRAY\_BUFFER

gl.ELEMENT\_ARRAY\_BUFFER

Vertex shader

A buffer



# Example (Ex02-1)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

`gl.bindBuffer(target, buffer)`

target: `gl.ARRAY_BUFFER` or `gl.ELEMENT_ARRAY_BUFFER`

<https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/bindBuffer>

`gl.ARRAY_BUFFER`

`gl.ELEMENT_ARRAY_BUFFER`

A buffer

Vertex shader



# Example (Ex02-1)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

`gl.bufferData(target, data, usage)`

target: `gl.ARRAY_BUFFER` or `gl.ELEMENT_ARRAY_BUFFER`  
usage: `gl.STATIC_DRAW`, `gl.STREAM_DRAW` or `gl.DYNAMIC_DRAW` (tell webgl how will you use the buffer. This only affects on performance)

<https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/bufferData>

`gl.ARRAY_BUFFER`

`gl.ELEMENT_ARRAY_BUFFER`

Vertex shader

[0.0, 0.5, 10.0  
-0.5, -0.5, 20.0,  
0.5, -0.5, 30.0]

# Example (Ex02-1)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);
  gl.enableVertexAttribArray(a_Position);

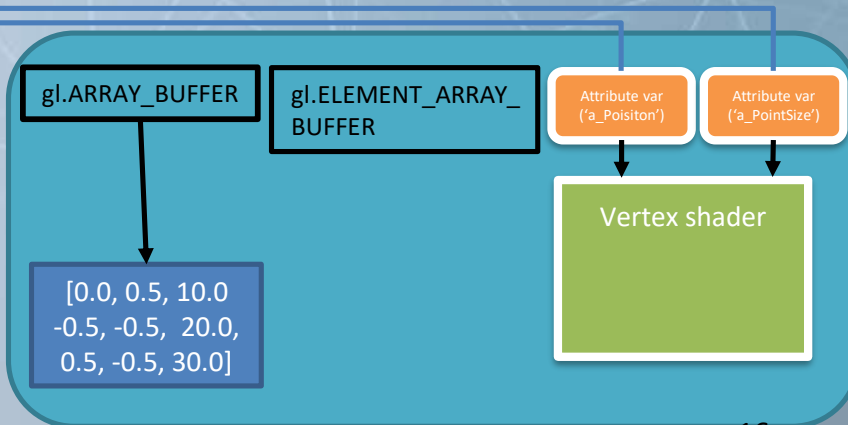
  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

`gl.getAttribLocation(program, varName):`

return reference of the attribute variable in vertex shader

<https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/getAttribLocation>



# Example (Ex02-1)

- WebGL.js: initVertex

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

`gl.vertexAttribPointer(location, size, type, normalized, stride, offset)`

location: where (attribute variable) to pass data

size: number of components per vertex shader attribute

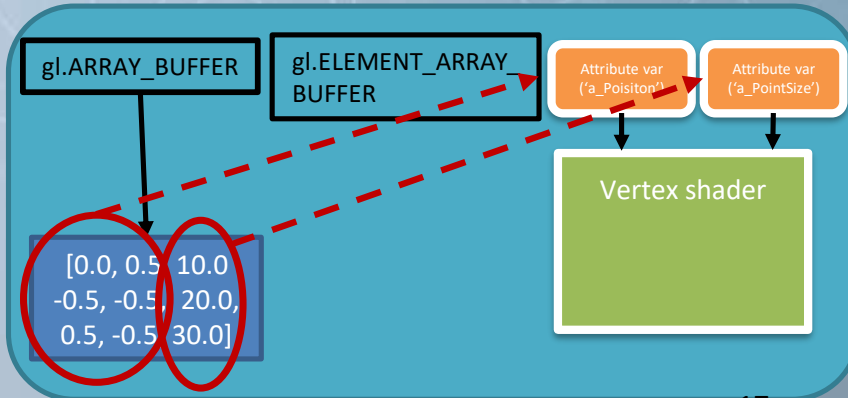
type: data type of each element

normalize: normalize data to a value range or not

stride: offset in bytes between the beginning of consecutive vertex attributes

offset: offset in bytes of the first component in the vertex attribute array

Document: <https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/vertexAttribPointer>



# Example (Ex02-1)

- WebGL.js: `initVertexBuffers()`

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE * 3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

4. Only pass two elements "a\_Position"


2. No offset for first element


1. an element is a "float"




3. Jump  $3 \times \text{FSIZE}$  bytes for attribute variable in next thread

GPU threads

 a\_Position = [0.0, 0.5]

 a\_Position = [-0.5, 0.5]

 a\_Position = [0.5, -0.5]

# Example (Ex02-1)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

  return n;
}
```

4. Only pass one element to "a\_PointSize"

2. Offset  $2 \times \text{FSIZE}$  bytes for the first element

3. Jump  $3 \times \text{FSIZE}$  bytes for attribute variable in next thread

1. an element is a "float"



GPU threads

 a\_PointSize = 10.0

 a\_PointSize = 20.0

 a\_PointSize = 30.0



# Example (Ex02-1)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 10.0,    //point0: x, y, size
     -0.5, -0.5, 20.0,  //point1: x, y, size
     0.5, -0.5, 30.0]   //point2: x, y, size
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE *3, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_PointSize = gl.getAttribLocation(program, 'a_PointSize');
  gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
  gl.enableVertexAttribArray(a_PointSize);

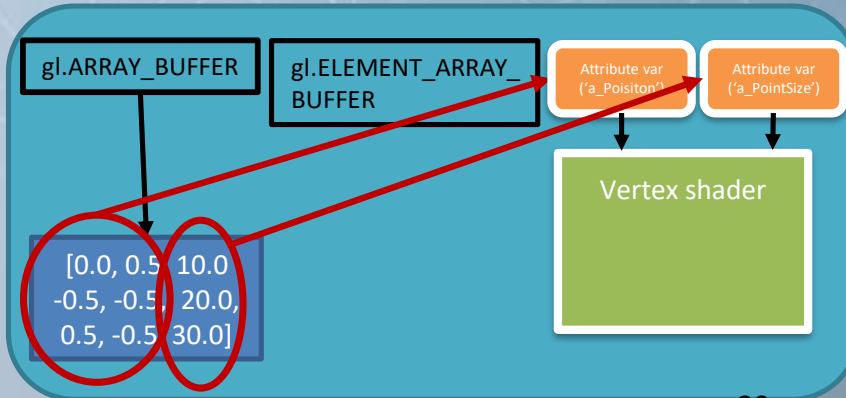
  return n;
}
```

You have to enable it to use

Return how many vertices to draw

`gl.enableVertexAttribArray(index)`

Document: <https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/enableVertexAttribArray>





# Example (Ex02-1)

- WebGL.js: main()

```
function main(){
  var canvas = document.getElementById('webgl');

  var gl = canvas.getContext('webgl2');
  if(!gl){
    console.log('Failed to get the rendering context for WebGL');
    return ;
  }

  let renderProgram = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

  gl.useProgram(renderProgram);

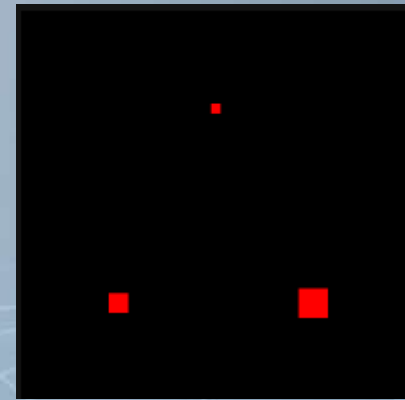
  var n = initVertexBuffers(gl, renderProgram);

  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);

  gl.drawArrays(gl.POINTS, 0, n);
}
```

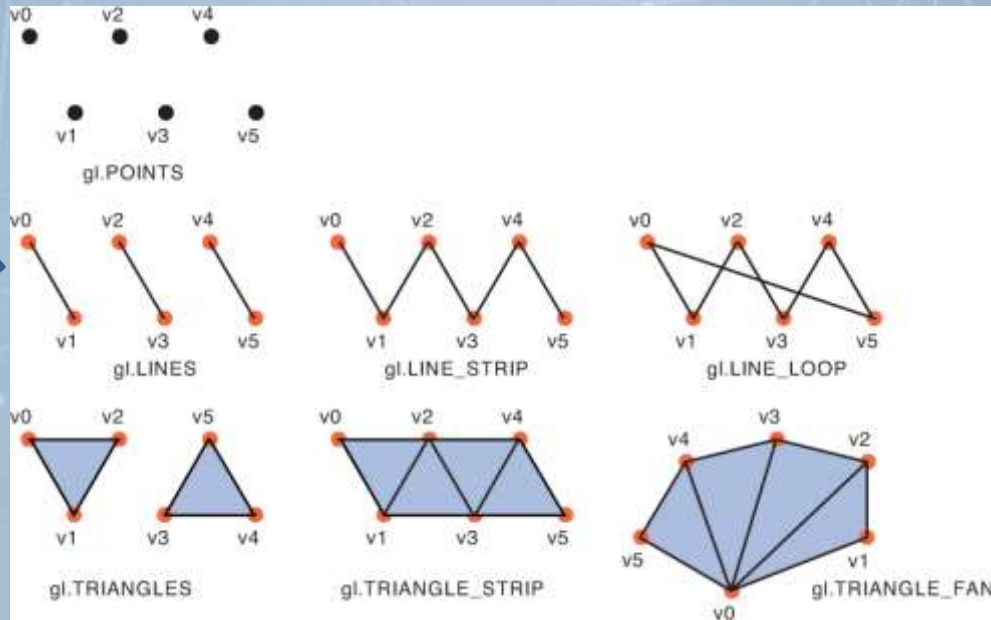
Clear screen by the background color

Call gl.drawArrays() once to run the shaders to draw



# gl.drawArrays

- `gl.drawArrays(mode, first, count);`
  - mode: define the connections between points (make surface)
  - first: the starting index in the array of vector points
  - count: number of index (vertex) to be drawn
- <https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/drawArrays>



# Let's Try

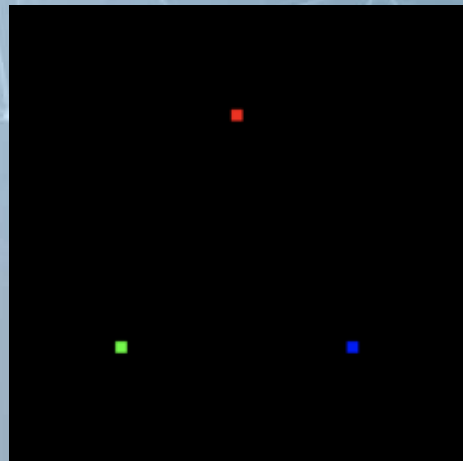
- Download Ex02-1 from Moodle
  - Try it on browser
- What you can try
  - If you comment Line88 and Line 93 (`gl.enableVertexAttribArray(a_Position);` and `gl.enableVertexAttribArray(a_PointSize);`), can it run?
  - If you change 'attribute' to 'uniform' in vertex shader, can it run?
  - What if you change the 'first' to 1 and 'count' to 2 in `gl.drawArrays`?
  - If you change 'gl.POINTS' at Line70 to
    - 'gl.LINES'
    - 'gl.LINE\_STRIP'
    - 'gl.LINE\_LOOP'
    - 'gl.TRIANGLES'

# “varying” in Shaders

- keywords for variables
  - “uniform”:
    - the variable in all threads have the same value.
    - Could be both in vertex or fragment shaders
  - “attribute”
    - The variable in all thread have different values
    - Could be only in vertex shader
  - “varying”
    - Declare “varying” variable in vertex and fragment shaders with the same name
      - the value will be passed **“from vertex shader to fragment shader”**
      - **“interpolation”** will be applied if necessary
    - e.g. varying vec3 color;

## Example (Ex:02-2)

- Draw three points with different colors
  - Pass color information from JavaScript to vertex shader
  - Then, pass the color information from vertex shader to fragment shader using “varying” variable
- Files
  - Index.html
  - WebGL.js



## Example (Ex:02-2)

- WebGL.js: shaders

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    varying vec4 v_Color;
    void main(){
        gl_Position = a_Position;
        gl_PointSize = 10.0;
        v_Color = a_Color;
    }
`;
```

```
var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

varying variable  
with the same name



# Example (Ex:02-2)

- WebGL.js: initVertexBuffers()

```
function initVertexBuffers(gl, program){
    var n = 3;

    var vertices = new Float32Array(
        [0.0, 0.5, 1.0, 0.0, 0.0, //point0: x, y, R, G, B
        -0.5, -0.5, 0.0, 1.0, 0.0, //point1: x, y, R, G, B
        0.5, -0.5, 0.0, 0.0, 1.0] //point2: x, y, R, G, B
    );

    var vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
    var FSIZE = vertices.BYTES_PER_ELEMENT;

    var a_Position = gl.getAttribLocation(program, 'a_Position');
    gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0);
    gl.enableVertexAttribArray(a_Position);

    var a_Color = gl.getAttribLocation(program, 'a_Color');
    gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2);
    gl.enableVertexAttribArray(a_Color);

    return n;
}
```

## Let's Try

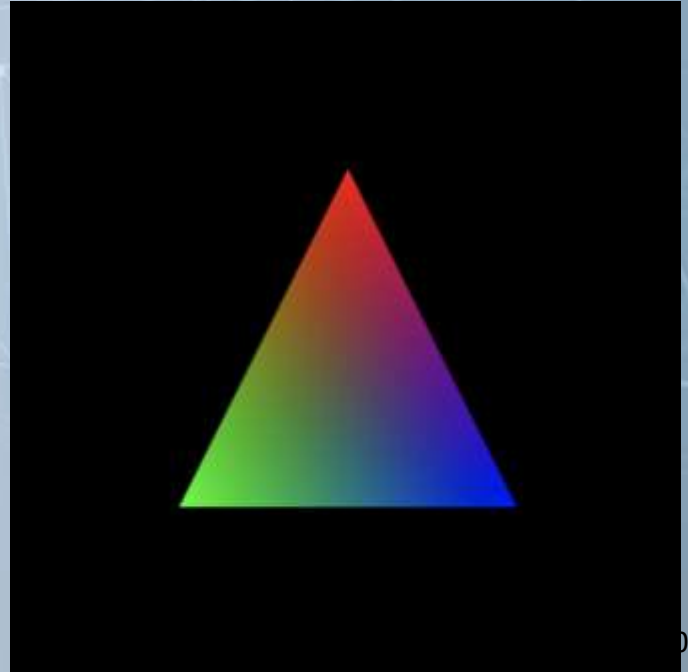
- Download Ex02-2 from Moodle
  - Try it on browser
  - What happen if you change 'gl.POINTS' at Line74 to 'gl.TRIANGLES'

# Interpolation

- When vertex shader pass “varying variable” to fragment shader, interpolation is applied within a surface

## Example (Ex:02-3)

- Draw a triangle with varying color
- Files
  - Index.html
  - WebGL.js



## Example (Ex:02-3)

- WebGL.js: main(), initVertexBuffer()

```
function main(){
  var canvas = document.getElementById('webgl');

  var gl = canvas.getContext('webgl2');
  if(!gl){
    console.log('Failed to get the rendering context for WebGL');
    return ;
  }

  let renderProgram = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

  gl.useProgram(renderProgram);

  var n = initVertexBuffers(gl, renderProgram);

  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);

  gl.drawArrays(gl.TRIANGLES, 0, n);
}
```

```
function initVertexBuffers(gl, program){
  var n = 3;

  var vertices = new Float32Array(
    [0.0, 0.5, 1.0, 0.0, 0.0, //point0: x, y, R, G, B
     -0.5, -0.5, 0.0, 1.0, 0.0, //point1: x, y, R, G, B
     0.5, -0.5, 0.0, 0.0, 1.0] //point2: x, y, R, G, B
  );

  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var FSIZE = vertices.BYTES_PER_ELEMENT;

  var a_Position = gl.getAttribLocation(program, 'a_Position');
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0);
  gl.enableVertexAttribArray(a_Position);

  var a_Color = gl.getAttribLocation(program, 'a_Color');
  gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2);
  gl.enableVertexAttribArray(a_Color);

  return n;
}
```

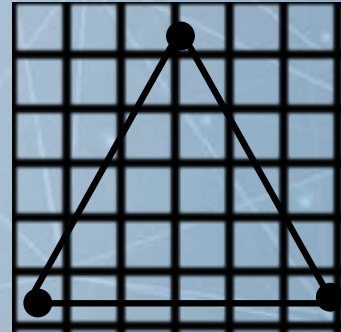
# Example (Ex:02-3)

- WebGL.js: shaders

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    varying vec4 v_Color;
    void main(){
        gl_Position = a_Position;
        v_Color = a_Color;
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

gl\_Position = [0.0, 0.5]  
varying v\_Color = [1.0, 0.0, 0.0]



gl\_Position = [-0.5, -0.5]  
varying v\_Color = [0.0, 1.0, 0.0]

gl\_Position = [0.5, -0.5]  
varying v\_Color = [0.0, 0.0, 1.0]



# Example (Ex:02-3)

- WebGL.js: shaders

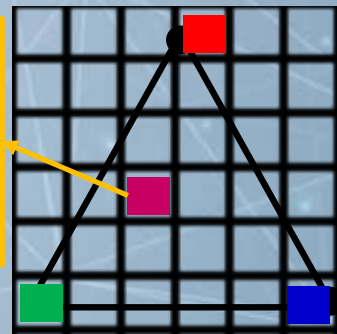
```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    varying vec4 v_Color;
    void main(){
        gl_Position = a_Position;
        v_Color = a_Color;
    }
`;
```

```
var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

This interpolation is done by WebGL automatically between vertex and fragment shader

gl\_Position = [0.0, 0.5]  
varying v\_Color = [1.0, 0.0, 0.0]

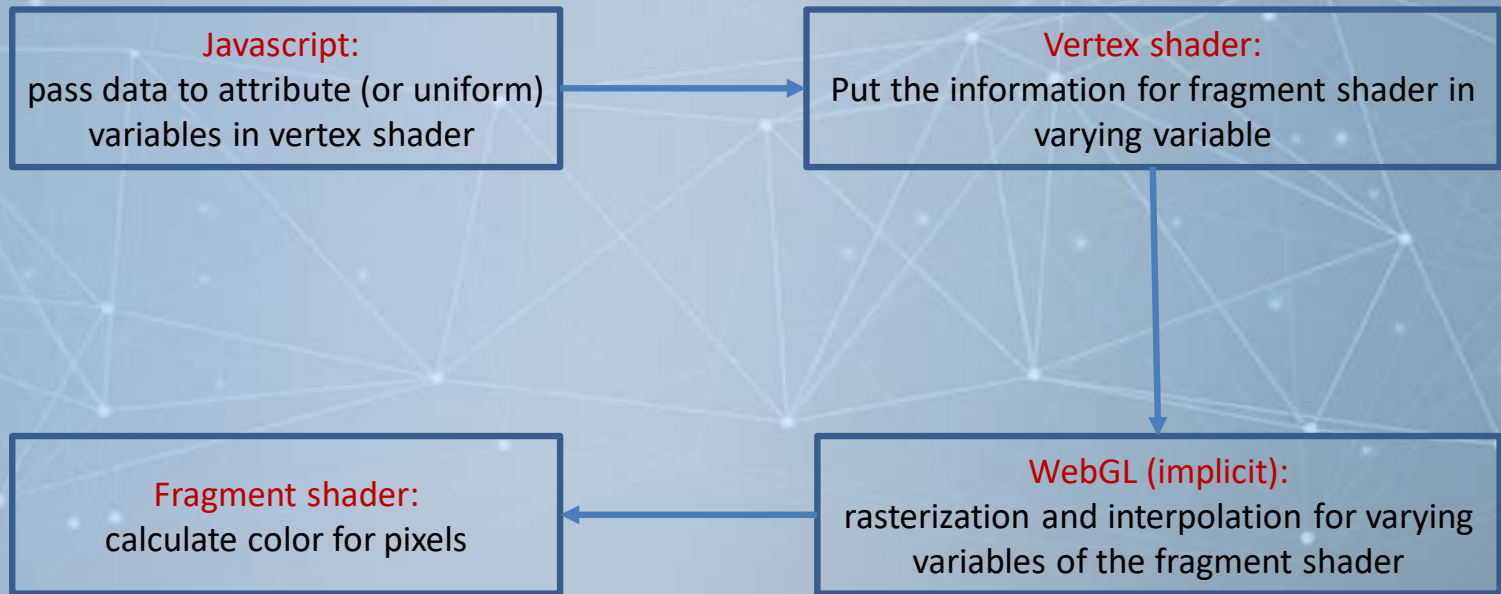
value of v\_Color in the thread of the fragment shader of this pixel is automatically calculated by the interpolation from the three vertices's v\_Color



gl\_Position = [-0.5, -0.5]  
varying v\_Color = [0.0, 1.0, 0.0]

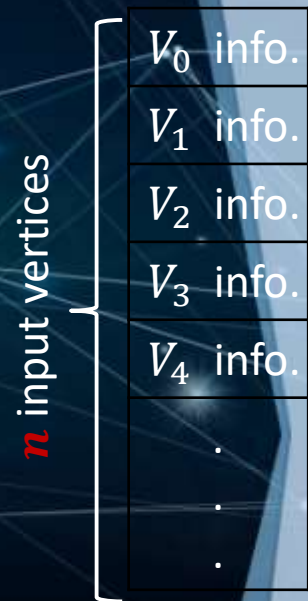
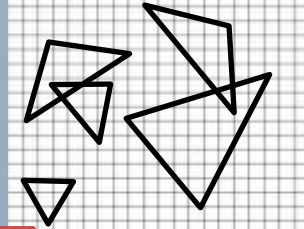
gl\_Position = [0.5, -0.5]  
varying v\_Color = [0.0, 0.0, 1.0]

# Summary of Vertex and Fragment Shader Pipeline



# How Many Copies Vertex and Fragment Shaders are produced to Parallely Process Your Input

Note: this is just for you to easily imagine how the shader runs (the number of copies of shaders may not be exactly the same as what happen on your machine because of some optimization may be applied)



$n$  copies of V-shaer to process vertices

V-Shader code

V-Shader code copy to run

V-Shader code copy to run

V-Shader code copy to run

.

.

.



Rasterization: WebGL automatically count all surfaces cover  $m$  screen pixels



$m$  copies of F-shaer to process vertices

F-Shader code

F-Shader code copy to run

f-Shader code copy to run

F-Shader code copy to run

F-Shader code copy to run

.

.

.

.

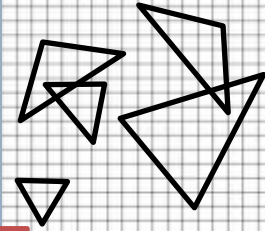


"Depth test" (deal with the pixel color occlusions)

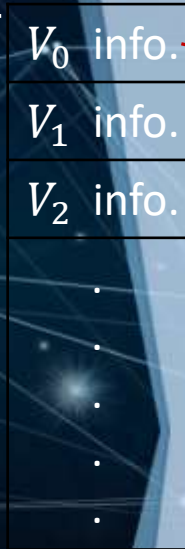
(The depth test may happen earlier, that is machine/version dependence)

# How Many Copies Vertex and Fragment Shaders are produced to Parallely Process Your Input

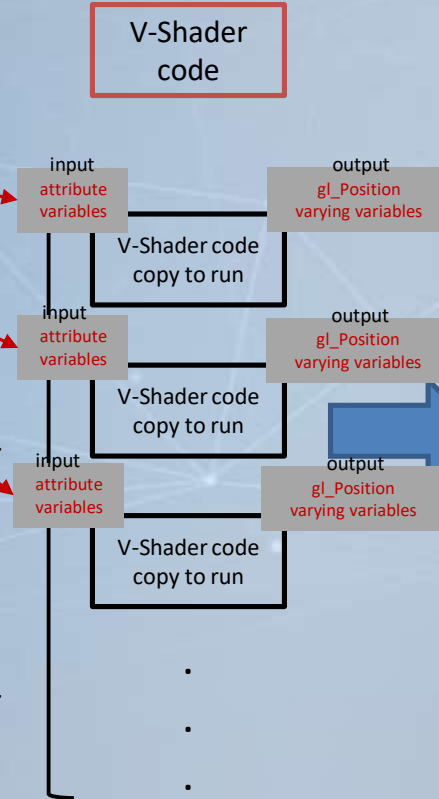
Note: this is just for you to easily imagine how the shader runs (the number of copies of shaders may not be exactly the same as what happen on your machine because of some optimization may be applied)



$n$  input vertices



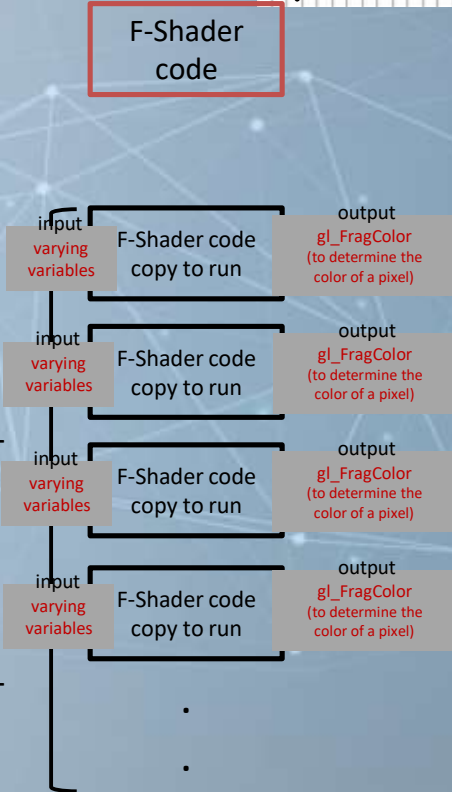
$n$  copies of V-shaer to process vertices



Rasterization: WebGL automatically count all surfaces cover  $m$  screen pixels



$m$  copies of F-shaer to process vertices



"Depth test" (deal with the pixel color occlusions)