# WebGL

Pierre, Jenny, and Xuepeng

# What we do before to implement 3D vision?

- 2D flash


# Can you do this with a 2D game?

real 3D -PAL5 with full view editor

# What we do before to implement 3D vision?

- 2D flash
- 2.5D flash

# Is it enough?

real 3d - Pandaria WOW (notice the waterfall)

# Is it enough？

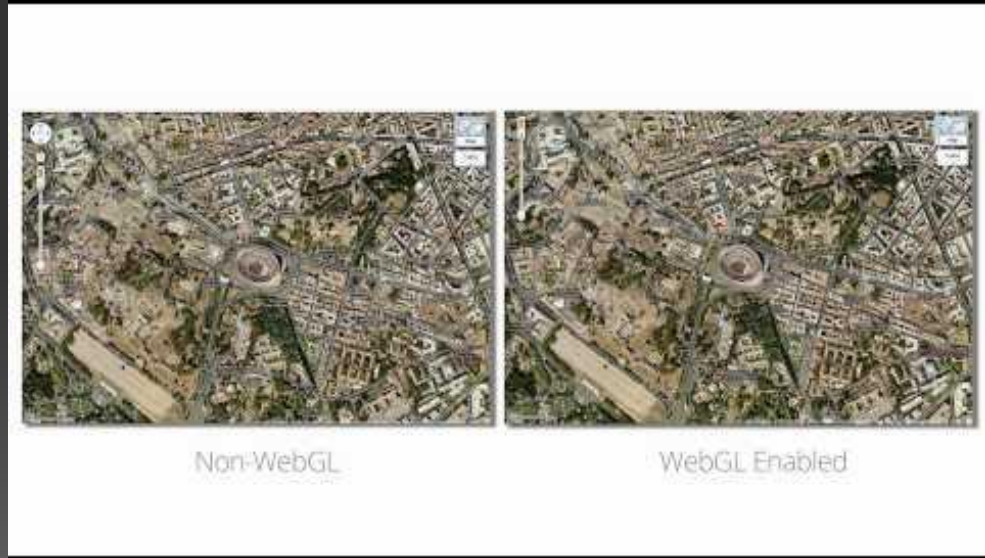

2.5d web game

So, we need to find a way to represent real 3D in web.

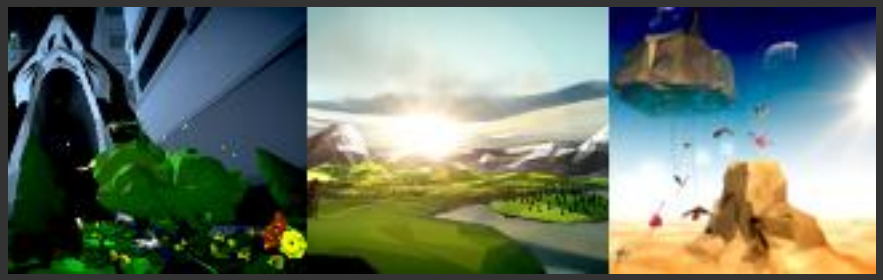
# Web GL

- 2006 started by Vladimir Vukićević at Mozilla
- 2007 both Mozilla and Opera had made their own separate implementations.
- 2009 Mozilla and Khronos started the WebGL Working Group
- 2011 Version 1.0 of the WebGL specification was released.


GWT Quake II
http://quake2-gwt-port.appspot.com/
Google
HTML5 Audio


Non-WebGL
WebGL Enabled

# Other examples



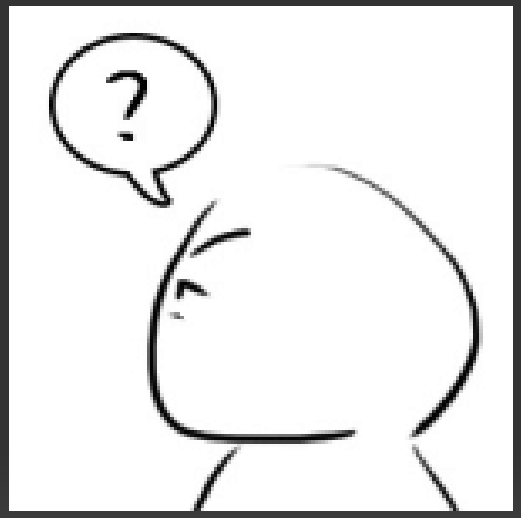
http://www.ro.me/

http://alteredqualia.com/three/examples/materials_shaders_fresnel.html

# What is WebGL?

WebGL is a context of the canvas HTML element that provides a 3D computer graphics API without the use of plug-ins.

canvas HTML element ??



# HTML 5

- the fifth version of the HTML standard
- what is new？
  - New elements: article, aside, audio and so on
  - New APIs：
    - The canvas element for immediate mode 2D drawing.
    - Timed media playback
    - Offline storage database (offline web applications)

# HTML5 history

- 2004 The Web Hypertext Application Technology Working Group (WHATWG)started.
- 2009 The W3C allowed the XHTML 2.0 Working Group's charter to expire and decided not to renew it.
- 2010 "thought of flash" by Steven Jobs
  - "Flash is no longer necessary to watch video or consume any kind of web content,new open standards created in the mobile era, such as HTML5, will win"

# HTML5 Graphics & Embedded Content

| | WIN | | | | | | | | | | MAC | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FIREFOX | | SAFARI | IE | | | | | CHROME | OPERA | FIREFOX | SAFARI | OPERA | |
| | 3.6 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 10 | 11.1 | 4 | 5 | 11.1 | |
| Canvas | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 87% |
| Canvas Text | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 85% |
| SVG | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 86% |
| SVG Clipping Paths | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 86% |
| SVG Inline | ✘ | ✔ | ✘ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ | 23% |
| SMIL | ✘ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 49% |
| WebGL | ✘ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 32% |
| Audio | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 85% |
| Video | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | 85% |

form: http://www.findmebyip.com/litmus

# Mobile supported

# What inside?

## OpenGL ES

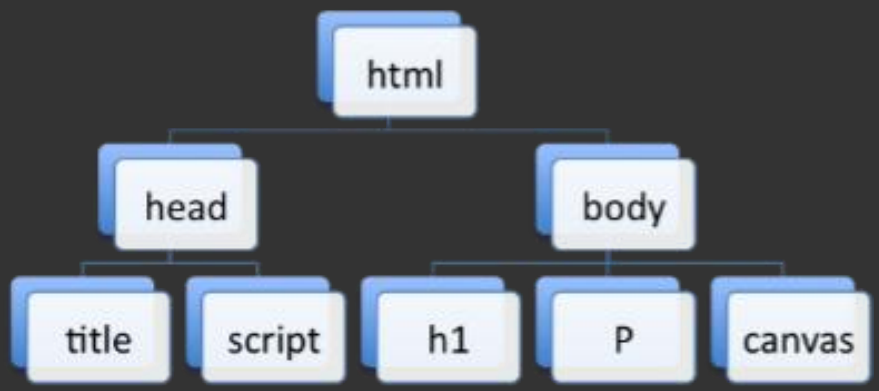
- is a subset of the OpenGL 3D graphics application programming interface (API) designed for embedded systems such as mobile phones, PDAs, and video game consoles.

# Document Object Model (DOM)

- DOM defines a standard for accessing documents like XML and HTML

- HTML DOM is
  - A standard object model for HTML
  - A standard programming interface for HTML
  - Platform- and language-independent
  - A W3C standard
- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

# HTML DOM




```
<html>
    <head>
        <title>Example</title>
        <script src="xxx.js" type="text/javascript">
        </script>
    </head>
    <body>
        <h1>webgl</h1>
        <p>canvas</p>
        <canvas id="example">not work</canvas>
    </body>
</html>
```

# Canvas

- Canvas was initially introduced by Apple for use inside their own Mac OS X WebKit component in 2004.
- The <canvas> tag is used to draw graphics, on the fly, via scripting (usually JavaScript).
- The <canvas> tag is only a container for graphics, you must use a script to actually draw the graphics.
- Canvas Rendering Context 2D
- WebGL Rendering Context

# Get access to HTML using API

- var canvas=document.getElementById('canvasID');
- *var gl=canvas.getContext('experimental-webgl');*
- gl is object of WebGL Rendering Context.
- first time the getContext called, a drawing buffer is created.
- Subsequent calls return the same object.
- There are more parameters for getContext to set the attributes of drawing buffer. such as alpha, depth, etc.
- for example, if you set alpha to false in the argument, there will no alpha channel in the color attributes.
- *var gl=canvas.getContext('experimental-webgl',{alpha:false});*

# A better way to initialize WebGL

- if fail to initialize WebGL, tell user why and how to solve.
  - browser doesn't support WebGL
  - some other reasons
- different browser is a little different on initializing
  - webgl
  - experimental-webgl
  - webkit-3d
  - moz-webgl

# A better way to initialize WebGL (Cont.)

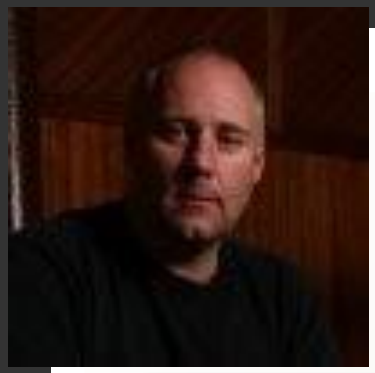
```
if(!window.WebGLRenderingContext){
    alert("Your browser doesn't support WebGL!");
    window.location="http://get.webgl.org";
}
else{
    var canvas=documents.getElementById("myCanvas");
    var names=["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];
    var gl=null;
    var for(var i=0;i<names.length;++i){
        try{
            gl=canvas.getContext(names[i],opt_attribs);
        }
        catch(e){
            if(gl){
                break;
            }
        }
}
```

# Two essential, Two optional for OpenGL

Essential:
Shader
Program

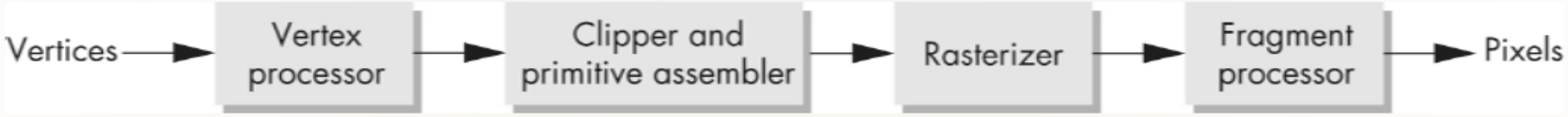
Optional:
VBO
VAO

# Let's take a little bit review....

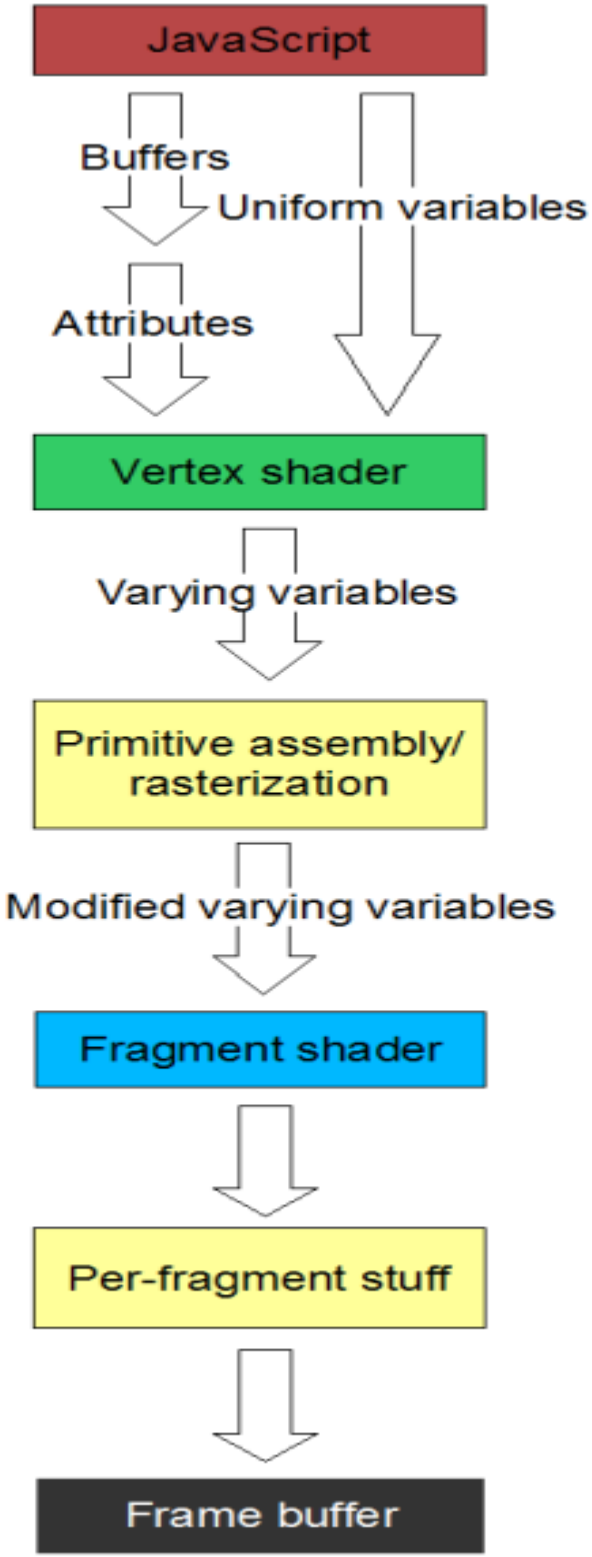


## Modern OpenGL

- Performance is achieved by using GPU rather than CPU
- Control GPU through programs called shaders
- Application's job is to send data to GPU
- GPU does all rendering



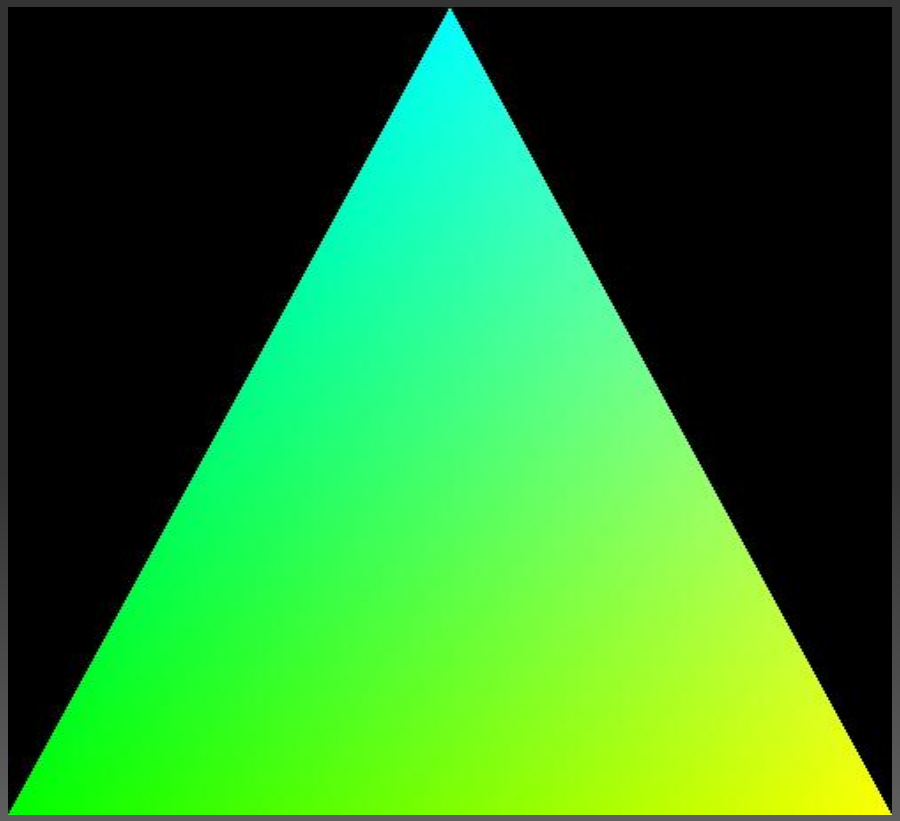

From CS 435 "Chapter02_-_Part_1_1-up_" slide #8




# Well, exactly the same thing here!

# Shader Initialization

The way WebGL initialize the shader is the same as OpenGL do. Shaders are written in GLSL. Shaders must be loaded with a source string, compiled, and then attached to a program.



# Writing Shaders

The way we write shaders in WebGL is similar to what we do in OpenGL.

Instead of writing shader in a text file, we give the shader a tag which tells javascript that the name of the shader and the type of the shader.

```
<script id = nameOfShader type = shaderType>
// GLSL in here.
</script>
```

Few things should keep in mind.....

# GLSL for WebGL

From WebGL Specification Version1.0
https://www.khronos.org/registry/webgl/specs/1.0/#SUPPORTED_GLSL_CONSTRUCTS

A WebGL implementation must only accept shaders which conform to The OpenGL ES Shading Language, Version 1.00. In particular, a shader referencing state variables or functions that are available in other versions of GLSL (such as that found in versions of OpenGL for the desktop), must not be allowed to load.

Identifiers starting with "webgl_" and "_webgl_" are reserved for use by WebGL. A shader which declares a function, variable, structure name, or structure field starting with these prefixes must not be allowed to load.

Some GLSL implementations disallow characters outside the ASCII range, even in comments. In such cases the WebGL implementation needs to prevent errors in such cases. The recommended technique is to preprocess the GLSL string, removing all comments, but maintaining the line numbering by inserting newline characters as needed for debugging purposes.

# Storage Qualifiers

## Storage Qualifiers [4.3]

Variable declarations may be preceded by one storage qualifier.

| | |
|---|---|
| *none* | (Default) local read/write memory, or input parameter |
| **const** | Compile-time constant, or read-only function parameter |
| **attribute** | Linkage between a vertex shader and OpenGL ES for per-vertex data |
| **uniform** | Value does not change across the primitive being processed, uniforms form the linkage between a shader, OpenGL ES, and the application |
| **varying** | Linkage between a vertex shader and fragment shader for interpolated data |

From WebGL Reference card
http://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

# Precision and Precision Qualifiers

## Precision and Precision Qualifiers [4.5]

Any floating point, integer, or sampler declaration can have the type preceded by one of these precision qualifiers:

| | |
|---|---|
| **highp** | Satisfies minimum requirements for the vertex language. Optional in the fragment language. |
| **mediump** | Satisfies minimum requirements for the fragment language. Its range and precision is between that provided by **lowp** and **highp**. |
| **lowp** | Range and precision can be less than **mediump**, but still represents all color values for any color channel. |

For example:

```
lowp float color;
varying mediump vec2 Coord;
lowp ivec2 foo(lowp mat3);
highp mat4 m;
```

Ranges & precisions for precision qualifiers (FP=floating point):

| | FP Range | FP Magnitude Range | FP Precision | Integer Range |
|---|---|---|---|---|
| **highp** | $(-2^{62}, 2^{62})$ | $(2^{-62}, 2^{62})$ | Relative $2^{-16}$ | $(-2^{16}, 2^{16})$ |
| **mediump** | $(-2^{14}, 2^{14})$ | $(2^{-14}, 2^{14})$ | Relative $2^{-10}$ | $(-2^{10}, 2^{10})$ |
| **lowp** | $(-2, 2)$ | $(2^{-8}, 2)$ | Absolute $2^{-8}$ | $(-2^{8}, 2^{8})$ |

A precision statement establishes a default precision qualifier for subsequent int, float, and sampler declarations, e.g.:

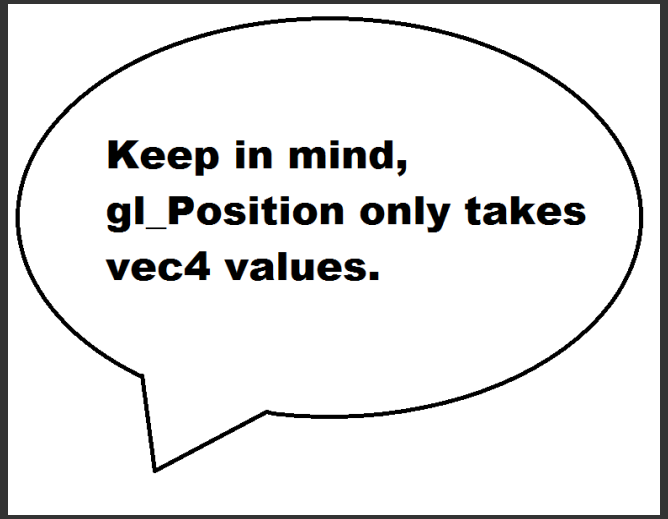
```
precision highp int;
```

Precision qualifiers has no affect in openGL, but you cannot do things without take care of it in OpenGL ES and WebGL !


From WebGL Reference card [http://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf](http://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf)

# Writing Shaders Example

```
<script id="vertexShader" type="x-shader/x-vertex">// Vertex shader
uniform mat4 modelViewProjMatrix;
    attribute vec4 vPosition;
    attribute vec4 vColor;
varying vec4 color;
void main() {
gl_Position = modelViewProjMatrix * vPosition;
color = vColor;
}
</script>
<script id="fragmentShader" type="x-shader/x-fragment">// Fragment shader
varying vec4 color;
void main(){
gl_FragColor = color;
}
</script>
```

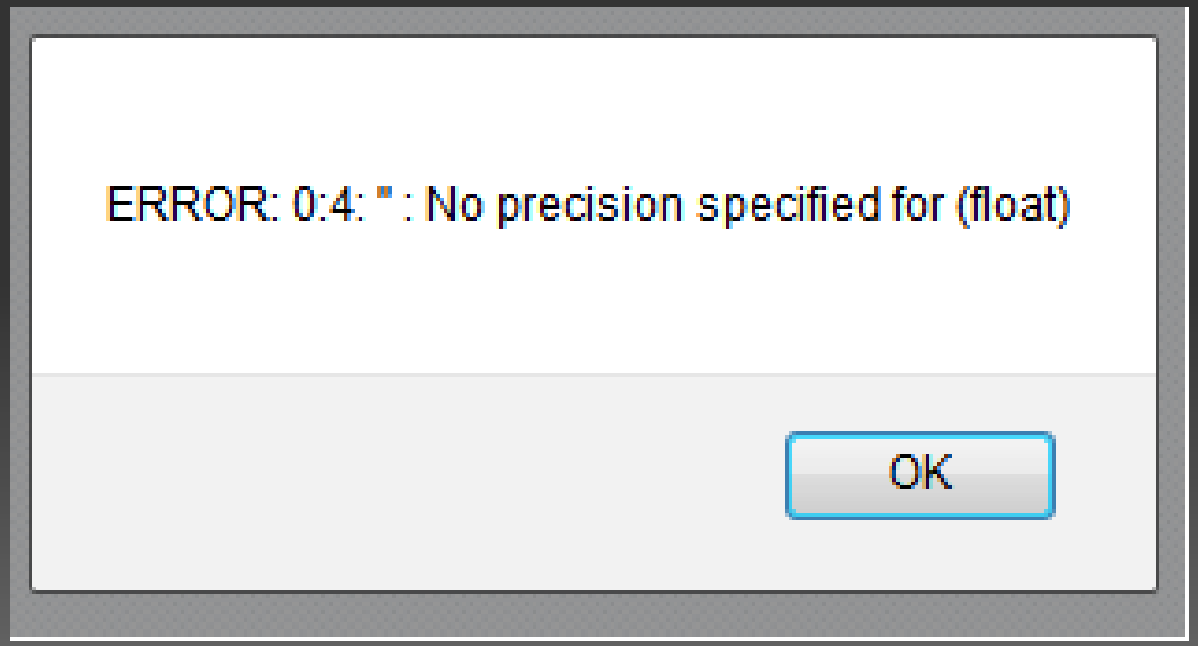


# Opps, got error

For fragment shader, it needs a bit of obligatory boilerplate code to tell the graphics card how precise we want it to be with floating-point numbers.

So, our fragment shader should looks like this:

```
<script id="fragmentShader" type="x-shader/x-fragment">
    #ifdef GL_ES
    precision highp float;
    #endif

varying vec4 color;
void main(){
gl_FragColor = color;
}
 </script>
```



# Load/Compile shaders

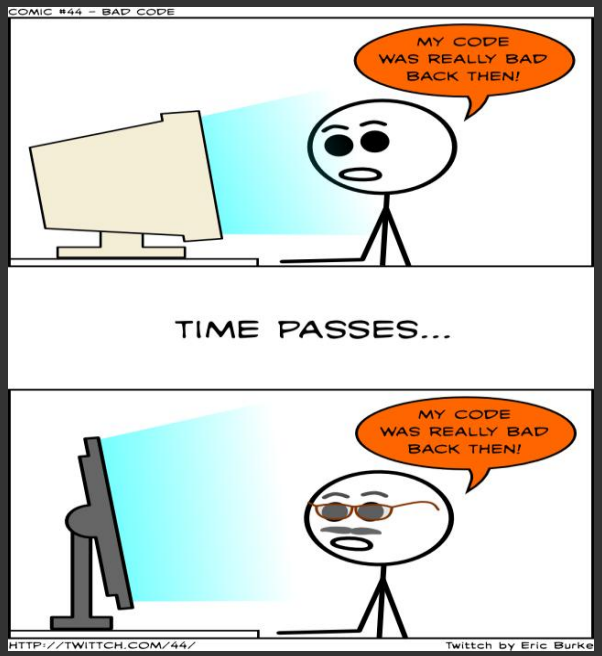
1. Find the tag of the shader script
   document.getElementById(shaderID);

2. Create shader object
   createShader(type_of_shader);

3. Load the source to the shader object
   shaderSource(shaderObject, shaderContent);

4. Compile it
   compileShader(shaderObject);

# Load/Compile shaders Example




```
function loadShader(ctx, shaderId) {
var shaderScript = document.getElementById(shaderId);
if (!shaderScript) {
 alter("*** Error: shader script '"+shaderId+"' not found");
 return null;
 }
// Create the shader object
var shader ;
if (shaderScript.type == "x-shader/x-vertex")
 shader = ctx.createShader(ctx.VERTEX_SHADER);
else if (shaderScript.type == "x-shader/x-fragment")
 shader = ctx.createShader(ctx.FRAGMENT_SHADER);
else {
 alter("*** Error: shader script '"+shaderId+"' of undefined type '"+shaderScript.type+"'");
 return null;
}
// Load the shader source
ctx.shaderSource(shader, shaderScript.text);
// Compile the shader
ctx.compileShader(shader);
return shader; }
```

Note: for the examples in this presentation, the word "ctx" or "gl" is the variable which stores the canvas context

# Check compile status

```
var compiled = ctx.getShaderParameter(shader, ctx.COMPILE_STATUS);
if (!compiled && !ctx.isContextLost())
{
  // Something went wrong during compilation; get the error
  var error = ctx.getShaderInfoLog(shader);
  alter("*** Error compiling shader '"+shaderId+"':"+error);
  ctx.deleteShader(shader);
 return null;
}
```

# Create/Attach/Link/Use Program

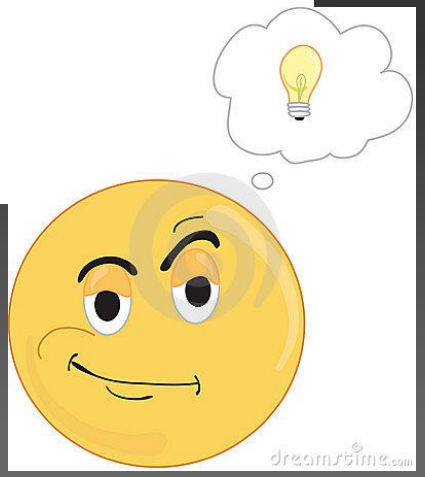

1. Get shaders

2. Create program object
   createProgram();

3. Attach shaders to program
   attachShader(programObject, shader);
4. Link/Use the program, setup background color, etc.
   linkProgram(programObject), useProgram(progranObject);
   clearColor(red, blue, green, alpha);

# Create/Attach/Link/Use Program Ex.

```
function initShaders(){
 var vertexShader = loadShader(gl, "vertexShader");
 var fragmentShader = loadShader(gl, "fragmentShader");

 program = gl.createProgram();
 gl.attachShader(program, vertexShader);
 gl.attachShader(program, fragmentShader);

 gl.linkProgram(program);

 if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
            alert("Could not initialise shaders");
        }
}
```

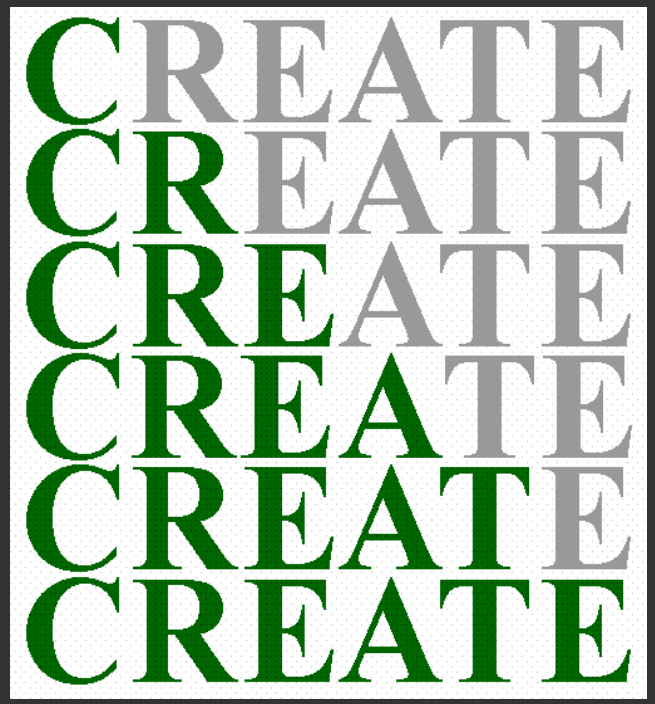


# WebGL: Sorry, we don't have VAO.

```
    return program;
}

// Initialize buffer object and vertex array object as well as the background
// colour and the shadrs. Also initialize the vertex position attribute from
// the vertex shader.
void init( void ) {
    // Get point array.
    vec2 *points = computePoints(1.0);
    int pointArraySize_ = numberOfPoint*sizeof(vec2);

    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );

    // Create and initialize a buffer object
    GLuint buffer;
    glGenBuffers( 1, &buffer );
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, pointArraySize_,
                  points, GL_STATIC_DRAW );

```

Since WebGL API does not support client-side arrays, we do not generate Vertex Array Object in our code.

# Create / Binding VBO




Well, still the same

1. Create VBO

   createBuffer();

2. Binding

   bindBuffer(enum target, bufferObject);

Where target can be one of ARRAY_BUFFER or ELEMENT_ARRAY_BUFFER. Once the buffer is bound to one of them, the binding point sticks with the buffer in its life time.

Notice: the target in OpenGL can be ARRAY_BUFFER, COPY_READ_BUFFER, COPY_WRITE_BUFFER, ELEMENT_ARRAY_BUFFER, PIXEL_PACK_BUFFER, PIXEL_UNPACK_BUFFER, TEXTURE_BUFFER, TRANSFORM_FEEDBACK_BUFFER, UNIFORM_BUFFER.

# After Binding… Fill The Buffer With Data

Remember the tip Stephen posted which talks about the parameter of glBufferData:(http://moodle.upei.ca/mod/forum/discuss.php?d=39552)

1. glbufferData is a method used with OpenGL that "creates and initializes a buffer object's data store". It takes four parameters [1]:

```
void glBufferData(
GLenum target,
GLsizeiptr size,
const GLvoid * data,
GLenum usage);
```

The main points can be read off the link. What I find interesting is the "**usage**" parameter. It gives the program information to access the buffered data intelligently, and can improve performance. It does not "constrain the actual usage of the data store". It is broken into the format:

"GL_(variable controlling frequency of access)_(variable controlling nature of access)"

Frequency of access variables[1]:

- **STREAM**: Contents modified once, used at most a few times.
- **STATIC**: Contents modified once, used many times.
- **DYNAMIC**: Contents modified repeatedly, used many times.

Nature of access variables[1]:

- **DRAW**: Contents modified by application, used as the source for GL drawing and image specification commands
- **READ**: Contents modified by reading data from the GL, and used to return that data when queried by the application.
- **COPY**: Contents are modified by reading data from the GL, and used as the source for GL drawing and image specification commands.

# WebGL do it in a different way

1. glbufferData is a method used with OpenGL that "creates and initializes a buffer object's data store". It takes four parameters [1]:

```
void glBufferData(
GLenum target,
GLsizeiptr size,
const GLvoid * data,
GLenum usage);
```

In WebGL, bufferData only has one of the two parameter

The main points can be read off the link. What I find interesting is the "**usage**" parameter. It gives the program information to access the buffered data intelligently, and can improve performance. It does not "constrain the actual usage of the data store". It is broken into the format:

"GL_(variable controlling frequency of access)_(variable controlling nature of access)"

Frequency of access variables[1]:

- **STREAM**: Contents modified once, used at most a few times.
- **STATIC**: Contents modified once, used many times.
- **DYNAMIC**: Contents modified repeatedly, used many times.

WebGL doesn't support these two

Nature of access variables[1]:

- **DRAW**: Contents modified by application, used as the source for GL drawing and image specification commands
- **READ**: Contents modified by reading data from the GL, and used to return that data when queried by the application.
- **COPY**: Contents are modified by reading data from the GL, and used as the source for GL drawing and image specification commands.

# WebGL do it in a different way(cont.)

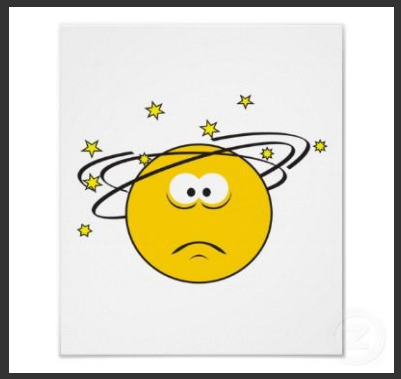
That is, if you want to fill the buffer with data, you need to call either bufferData(target, data, usage) or bufferData(target, dataSize, usage).

And for usage, you only have three choices:
STREAM_DRAW
STATIC_DRAW
DYNAMIC_DRAW

Similar change in bufferSubData:
For OpenGL: glBufferSubData(target, offset, size, dataObject)

For WebGL: bufferSubData(target, offset, dataObject)

# Example of VBO



```
var squareVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
var vertices = [   1.0, 1.0, 0.0,
                  -1.0, 1.0, 0.0,
                   1.0, -1.0, 0.0,
                  -1.0, -1.0, 0.0 ];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
squareVertexPositionBuffer.itemSize = 3;

squareVertexPositionBuffer.numItems = 4;
```

Note: WebGL doesn't support double-precision floating-point. That is, GL_DOUBLE (or Float64Array in this case) is useless.

# Sending Data To Shaders

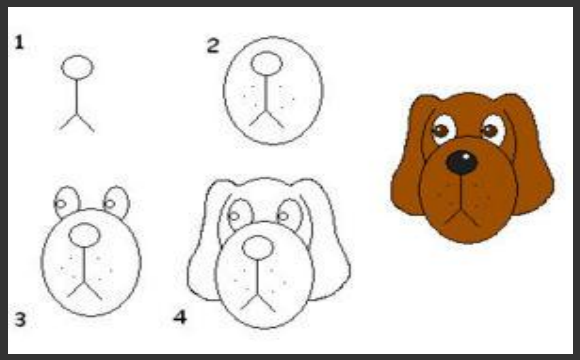

1. Get location of the attribute
   getAttribLocation(programObject, nameOfAttribute);

2. Enable the attribute array
   enableVertexAttribArray(locationOfTheAttribute);

3. Send data to shader
   vertexAttribPointer(locationOfTheAttribute, sizeOfEachItem, typeOfItem, normalized?, stride, offset);

Doing exactly same thing to send uniform data to shader:
   getUniformLocation(programObject, nameOfUniform);
   uniformMatrix[234]fv(locationOfUniform, transpose?, dataArray)

# How to draw




- Almost as same as OpenGL 3.1
- gl.drawArrays(GLenum mode,GLint first,GLsizei count);
- gl.drawElements(GLenum mode, GLsizei count, GLenum type,GLintptr offset);
- WebGL is much more safe.
- In drawElements, even the given offset is in bytes, it must be a valid multiple of size of the given type or an INVALID_OPERATION error will be genereated.
- as same as vertexAttribPointer
- if vertex attribute is used and has no buffer bound to it, the calls to drawArrays or draw will generate an INVALID_OPERATION error
- same thing for out of Range

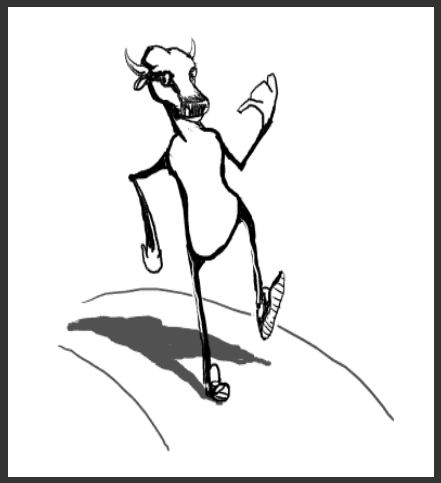
# Handling Lost Context


- Context can be lost when the GPU is not available for your program.
- Example:
  - another page does something take GPU too long and OS decides to reset the GPU to get the control back.
  - 2 or more pages use too many resources and the browser decieds to tell all the pages they lost the context and restore it only to the front page for now.
  - switches graphics cards
  - update the graphics driver

# Handling Lost Context (cont.)

- Events:
  - webglcontextlost
  - webglcontextrestored
  - webcontextcreationerror
- We can listen to these events and handle them
- addlistener("webglcontextlost",functionToHandleLost,false);
- addlistener("webglcontextstored", functionToHandleRestored,false);
- Once the context is restored, WebGL resources such as textures and buffers that were created before the context was lost are no longer valid. Reinitialize the context's state and resources.

# How to draw animation



- OpenGL 3.1: glutDisplayFunc(display);
- Animation is implemented by rendering loop.
- Browsers provide you API
- APIs are different between major browsers
  - webkitRequestAnimationFrame: Chrome, Safari
  - mozRequestAnimationFrame: firefox
  - msRequestAnimationFrame: IE
- Another way to draw animation is to use setTimeout function in JavaScript
- setTimeout heavily delay compared with RequestAnimationFrame

# How to draw animation (cont.)

- We should write a program to support all major browsers.
- Example:

```
window.requestAnimFrame = (function() {
    return window.requestAnimationFrame ||
            window.webkitRequestAnimationFrame ||
            window.mozRequestAnimationFrame ||
            window.oRequestAnimationFrame ||
            window.msRequestAnimationFrame ||
    function(/* function FrameRequestCallback */
callback,                          /* DOMElement Element */ element)
            {
                return window.setTimeout(callback, 1000/60);
            };
 })();
```

- window.requestAnimFrame(drawFunction);

# How to draw animation (cont.)

- Attention: we need to cancel the rendering loop when the context lost.

```
window.cancelRequestAnimFrame =
(function() {
 return window.cancelCancelRequestAnimationFrame ||
        window.webkitCancelRequestAnimationFrame ||
        window.mozCancelRequestAnimationFrame ||
        window.oCancelRequestAnimationFrame ||
        window.msCancelRequestAnimationFrame ||
        window.clearTimeout;
        })();
```

# Events

- There is actually no mouse events and keyboard events in WebGL
- Events in JavaScript is used for interaction
- canvas.onclick=clickFunction;
- canvas.onmouseup=mouseupFunction;

# Reference:

- http://learningwebgl.com/blog/
- https://www.khronos.org/registry/webgl/specs/1.0/#6.1
- http://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf
- http://www.khronos.org/webgl/wiki/Main_Page

# Question ?

# Thank you!

谢谢!