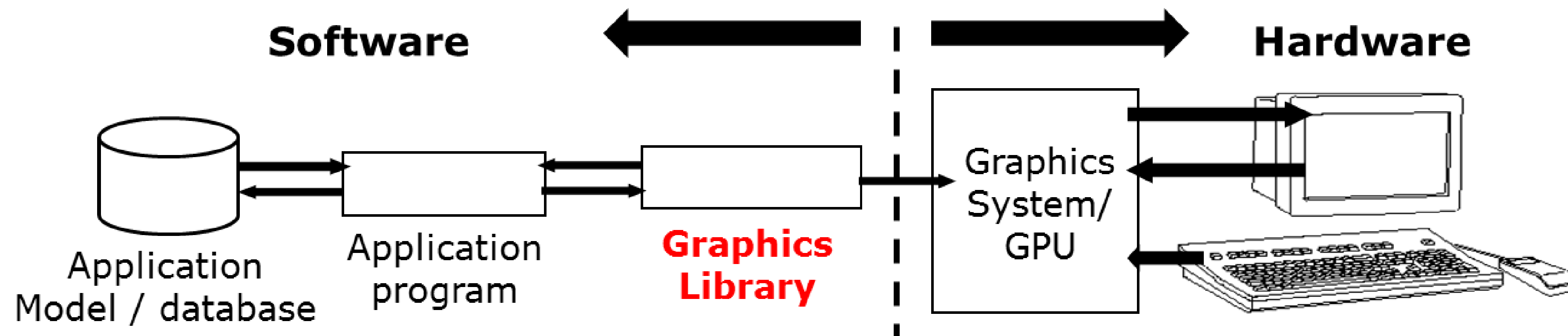
WebGL
A quick introduction

J. Madeira – V. 0.2 – September 2017

Interactive Computer Graphics

- Graphics library / package is intermediary between application and display hardware
- Application program maps / renders objects / models to images by calling on the graphics library
- User interaction allows image and / or model modification



[van Dam]

Graphics Library

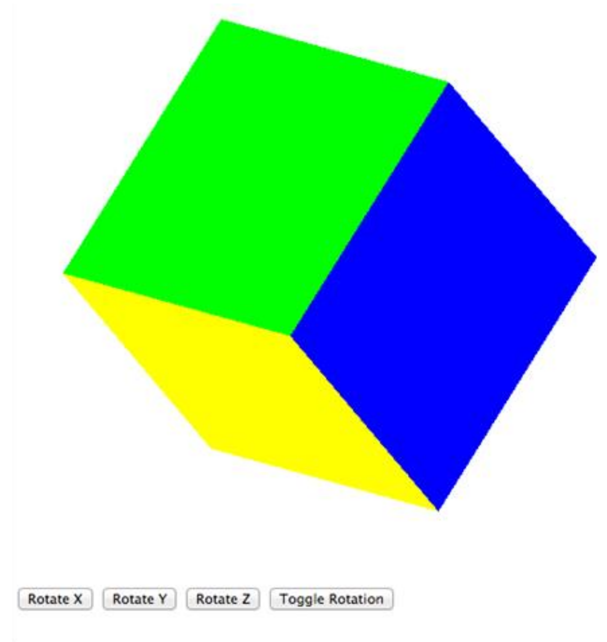
- Examples: OpenGL, RenderMan, DirectX, Windows Presentation Foundation (WPF), HTML5 + WebGL, …

- Primitives : characters, points, lines, triangles, …
- Attributes : color, line /polygon style, …
- Transformations : rotation, scaling, …
- Light sources
- Viewing
- …

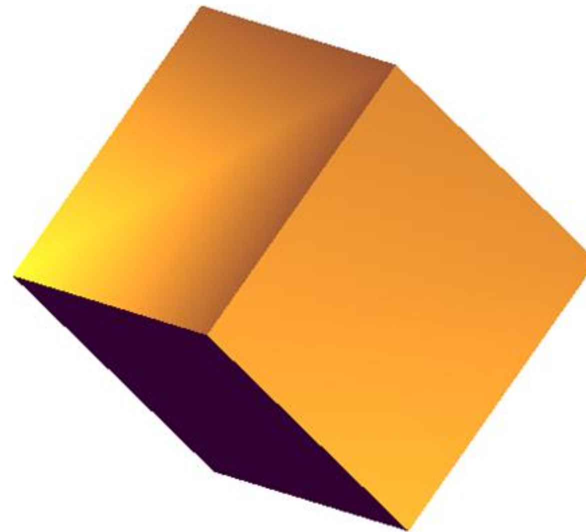
WebGL – Web Graphics Library

- JavaScript API
  - Operating system and windows system independent !
- Rendering interactive 2D and 3D Computer Graphics
  - Using local hardware
- Within any compatible web browser
  - No plug-ins necessary!
- Complete integration
  - GPU
  - Web page canvas (HTML5)
  - Mixing / compositing with other HTML elements
  - Integrates with standard Web packages and apps

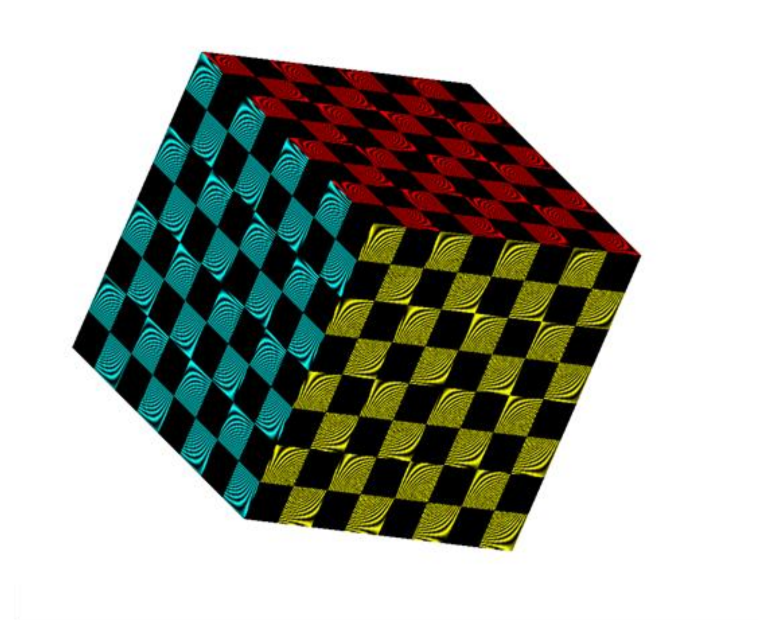
WebGL – Ed Angel's simple examples



rotating cube with buttons



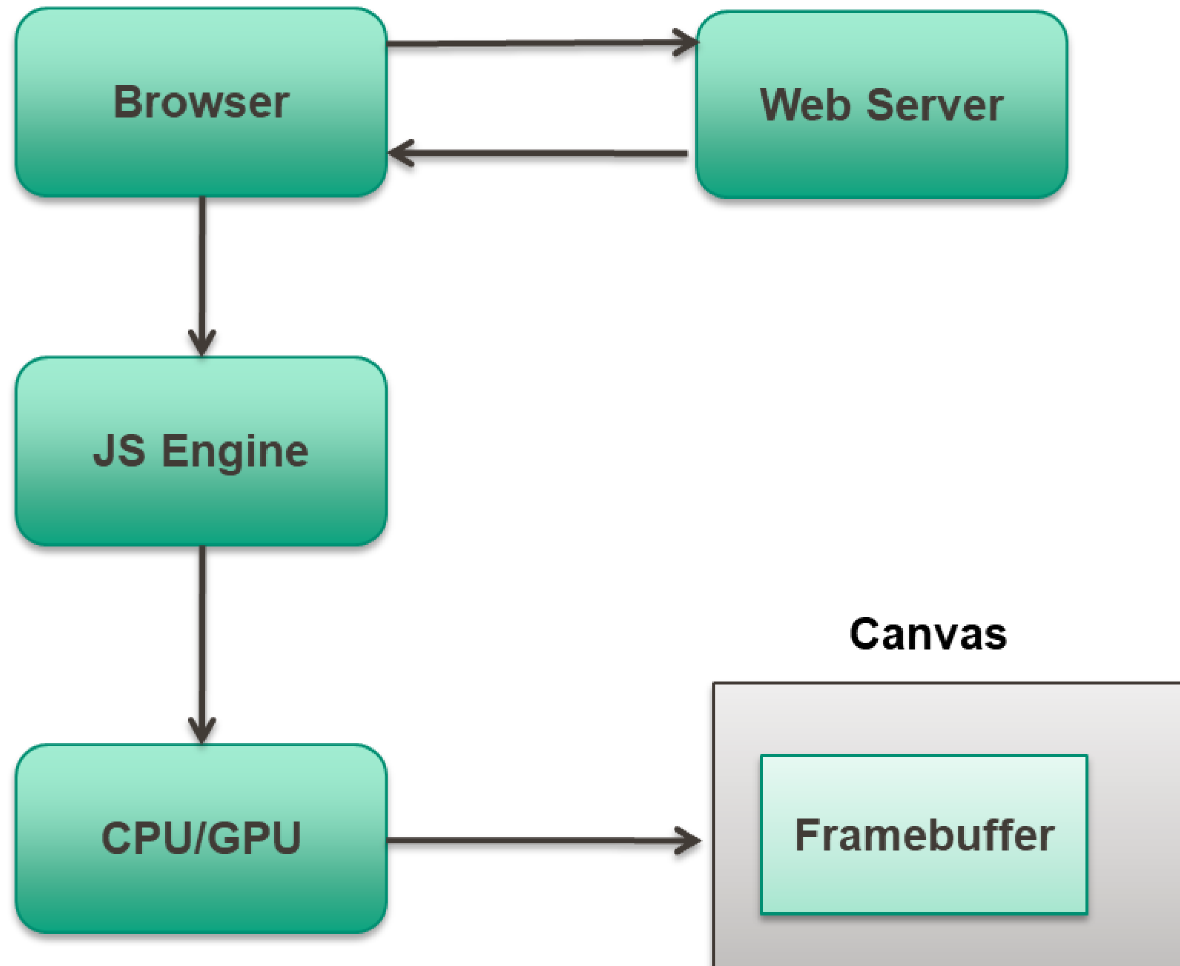
cube with lighting



texture mapped cube

[Angel / Shreiner]

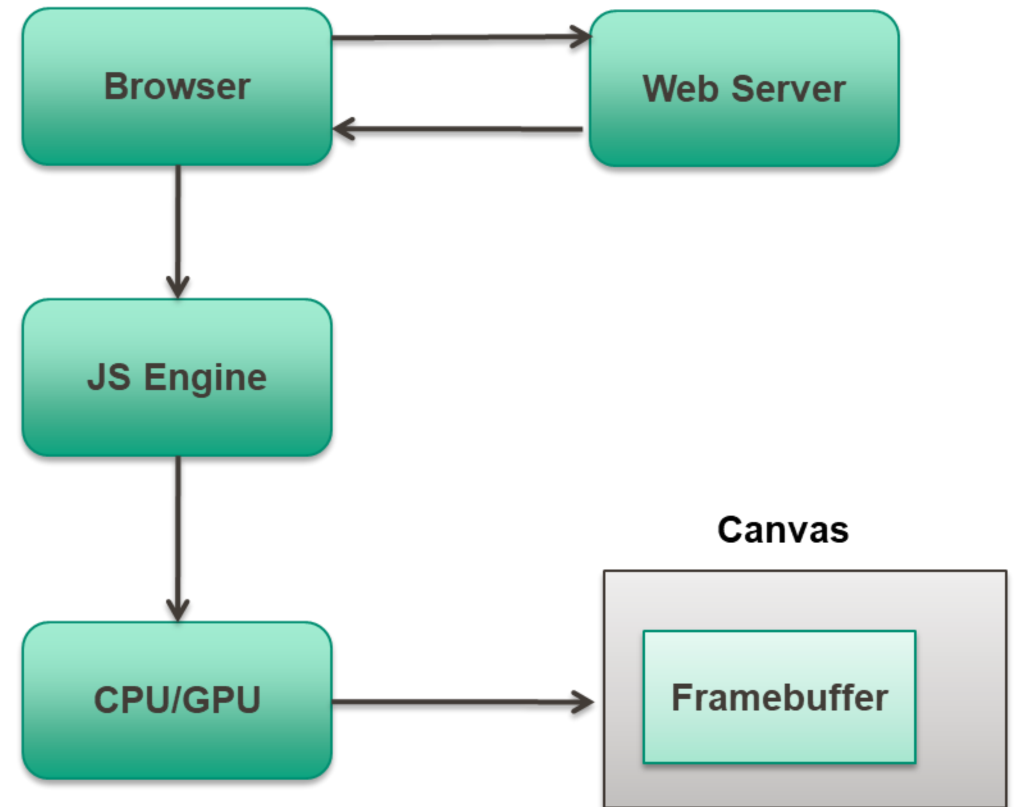
WebGL – Execution in browser



[Angel/Shreiner]

WebGL – Execution in browser

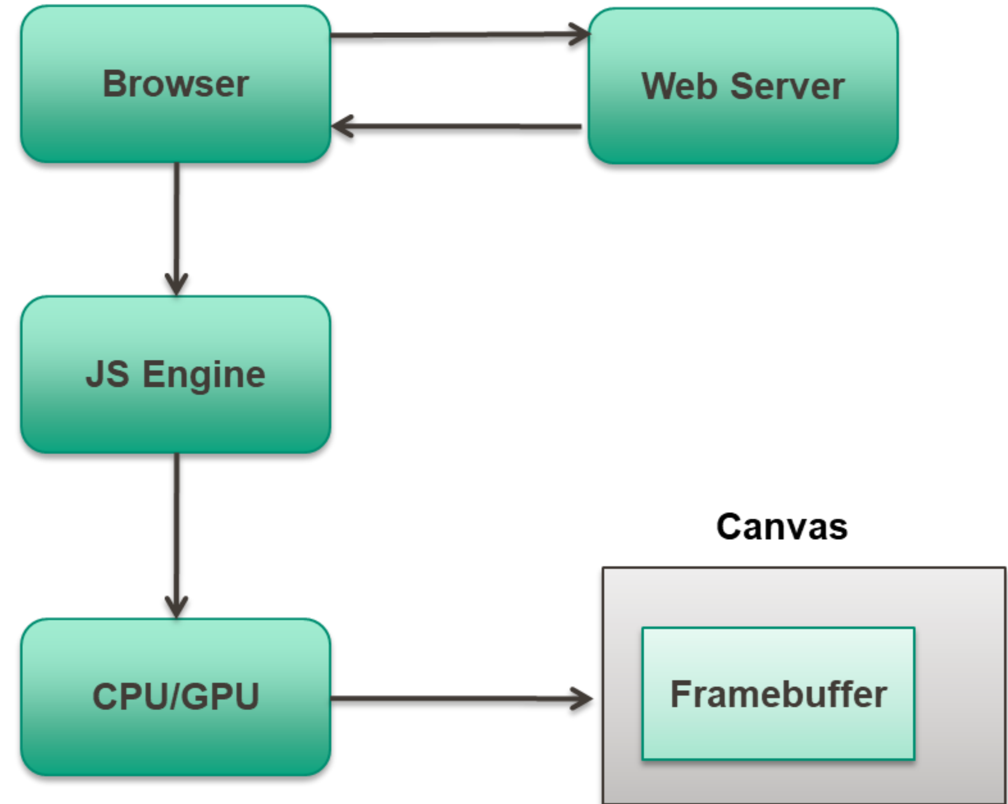
- Run WebGL on any recent browser
  - Chrome / Edge / Firefox / IE / Safari
- Code written in JavaScript
- JavaScript runs within browser
  - Use local resources



WebGL – Programs

- Control code – CPU
  - JavaScript
  - Send data to the GPU
  - Render function
    - Static application : execute once
    - Dynamic application : redraw after trigger events

- Shader code – GPU
  - GLSL
  - C / C++ like
  - Where the action happens !



WebGL – Programs

- HTML file
  - Describes Web page: structure, style and contents
  - Includes JS utilities and the JS application
  - Includes shader programs

- JavaScript files
  - Graphics
  - Modeling
  - Simulation

JavaScript

- JS is the language of the Web
  - Interpreted OO language
  - JS code executed in all browsers
- Interaction with the DOM
- Is JS slow?
  - JS engines are getting much faster
  - Not a key issue for graphics !
    - GPU handles the graphics data
- Use only a JS sub-set !

JavaScript

- Dynamic typing
- Scoping is different from most APIs
  - Watch out for globals
- Comparison operators: == vs === and != vs !===
- JS arrays are objects !
  - Not the same as in C / C++ / Java
  - WebGL expects C-style arrays

JavaScript – Arrays

- JS arrays are objects
  - Attributes: length, …
  - Methods: pop(), push(), shift(), unshift(), …
  - Dynamic resizing
  - WebGL expects C-style arrays
- JS typed arrays are like C arrays
  - Work with standard JS arrays
  - BUT, convert to typed arrays when sending data to the GPU
  - Use a flatten() function to extract data from a JS array object

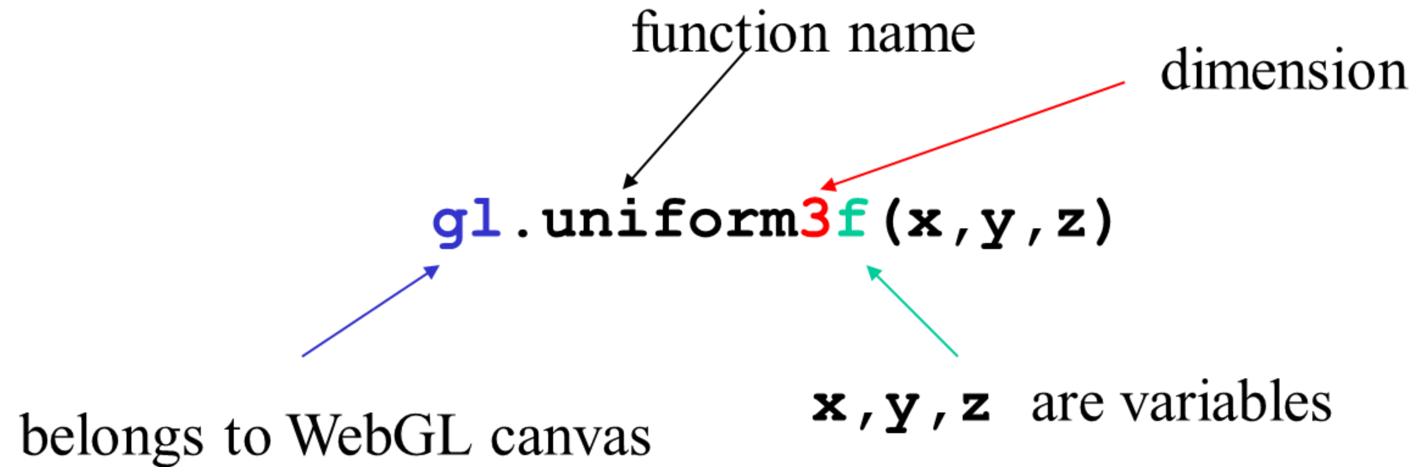
Minimalist approach

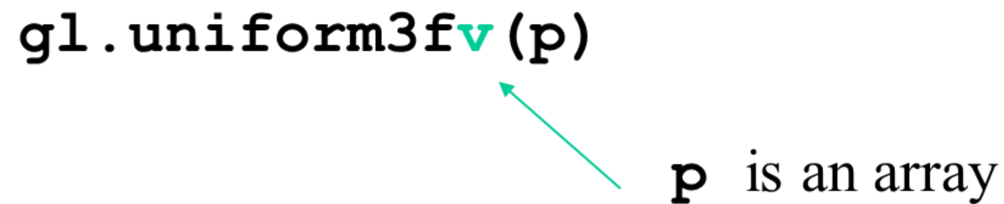
- Use only core JS and HTML
- No additional packages
- Focus on graphics
- If you want, you can be more ambitious !

WebGL – Some features

- Lack of object-orientation !
  - Multiple functions for the same logical function
  - Example
    - gl.uniform3f
    - gl.uniform2i
    - gl.uniform3dv

WebGL – Function format





[Angel]

WebGL – General structure

- Describe page (HTML file)
  - Request WebGL canvas
  - Read in necessary files
- Define shaders (HTML file)
  - Can be done with a separate file (browser dependent)
- Compute or specify data (JS file)
- Send data to GPU (JS file)
- Render data (JS file)

WebGL - Interaction

- Event-driven input uses callback functions or event listeners
- Define a callback function for each recognized event
- Browser enters an event loop and waits for an event
  - Buttons / Menus / Keyboard / Mouse
- It responds to the events for which it has registered callbacks
- The callback function is executed when the event occurs

HTML – onload event

- What happens after all files have been read?
  - HTML + JS + GLSL
- Use the onload event to initiate execution of the WebGL initialization function
  - onload event occurs when all files read

HTML – Buttons

```
<button id="button_1"> Change Color </button>
```

- Use HTML button tag for default style buttons
- id gives an identifier JS can use
- Text is displayed in the button
- Clicking on the button generates a click event
- Use CSS or jQuery to get prettier buttons

JS – Button-event listener

- Do not forget to define the listener
  - Otherwise the event occurs and is ignored
  - Two possibilities

```
var myButton = document.getElementById("button_1");

myButton.addEventListener("click", function() {
    …
});
```

```
document.getElementById("button_1").onclick =
function() { … };
```

HTML – Menus

```
<select id="mymenu" size=“2">
<option value="0">Item 1</option>
<option value="1">Item 2</option>
</select>
```

- Use the HTML select element
- Each menu entry is an option element
- With an integer value returned by a click event

JS – Menu listener

```
var m = document.getElementById("mymenu");
m.addEventListener("click", function() {
  switch (m.selectedIndex) {
    case 0:
      …
      break;
    case 1:
      …
      break;
   }
});
```

WebGL – General structure

- Describe page (HTML file)
  - Request WebGL canvas
  - Read in necessary files
- Define shaders (HTML file)
  - Can be done with a separate file (browser dependent)
- Compute or specify data (JS file)
- Send data to GPU (JS file)
- Render data (JS file)

GLSL – OpenGL Shading Language

- C / C++ like
  - Matrix and vector types (2D, 3D and 4D)
  - Overloaded operators
  - C++ like constructors

- Code sent to shaders as source code

- WebGL
  - Compile and link GLSL code
  - Send information / data to shaders

The simplest Vertex Shader

```
<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;

void main( void )
{
    gl_Position = vPosition;
}

</script>
```

The simplest Fragment Shader

```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

void main( void )
{
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}

</script>
```

Shaders

- Shaders are full programs
- Each shader has an id that can be used by JS code
- Shaders must set the two required built-in variables
  - `gl_Position`
  - `gl_FragColor`
- Must set precision in fragment shader

GLSL qualifiers

- Need qualifiers due to the nature of the execution model

- Variables can change (at most)
  - Once per primitive
  - Once per vertex
  - Once per fragment
  - At any time in the application

```
uniform   vec3 color;
attribute vec4 vPosition;
varying   vec3 fColor;
```

- Vertex attributes are interpolated by the rasterizer into fragment attributes

Shaders

```
//  Load shaders

var program = initShaders( gl, "vertex-shader",
                               "fragment-shader" );
gl.useProgram( program );

// Load the data into the GPU

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten( vertices ),
               gl.STATIC_DRAW );
```

Shaders

```
// Associate out shader variables with our data buffer

var vPosition = gl.getAttribLocation( program,
                                      "vPosition" );

gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false,
                        0, 0 );

gl.enableVertexAttribArray( vPosition );
```

Shaders

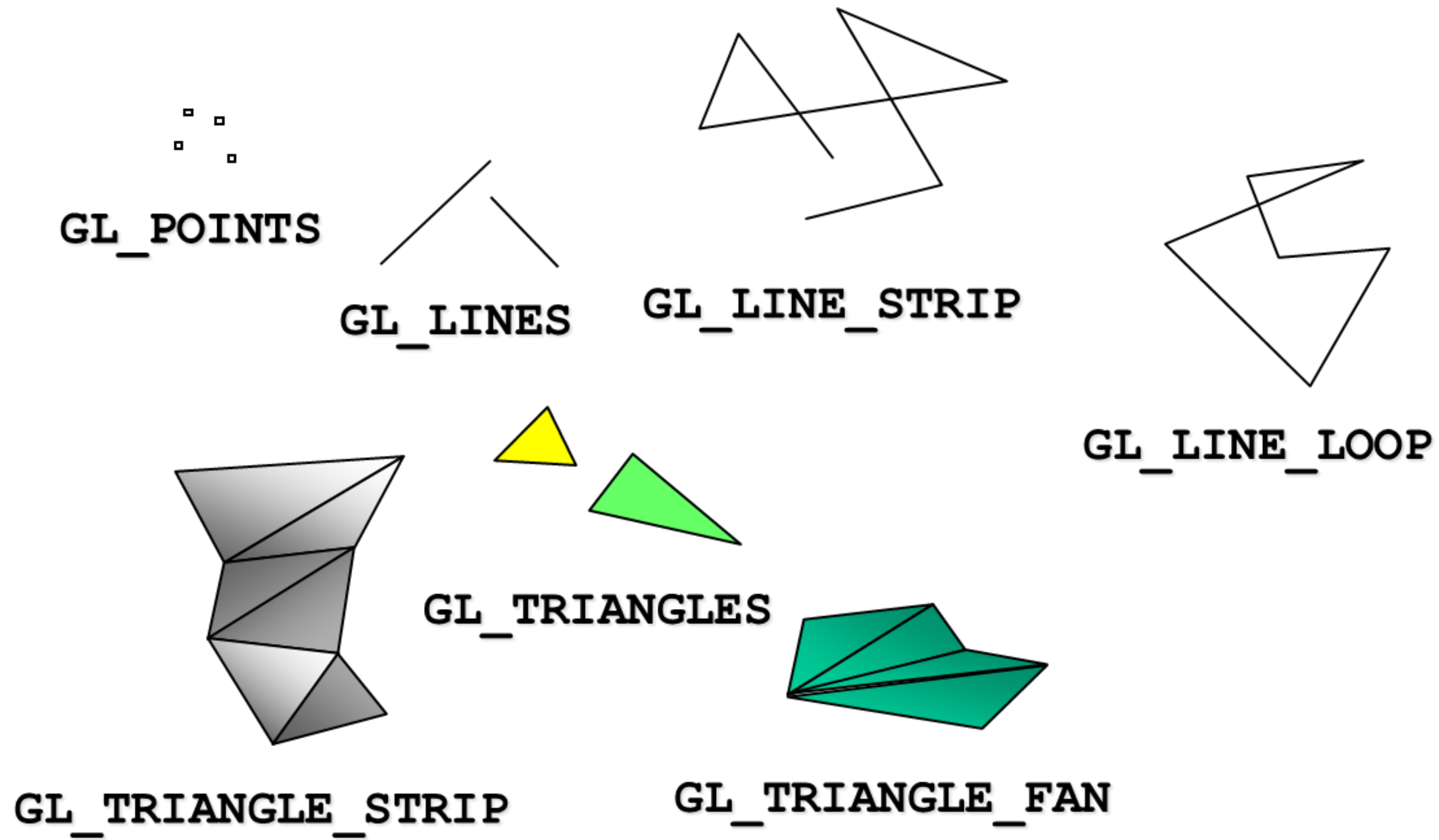
- initShaders used to load, compile and link shaders to form a program object
- Load data on the GPU by creating a vertex buffer object on the GPU
  - Use flatten() to convert the JS array to an array of floats
- Connect JS variables with shader variables
  - Need name, type and location in buffer

How to render?

```
function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.LINES, 0, 4 );
}
```

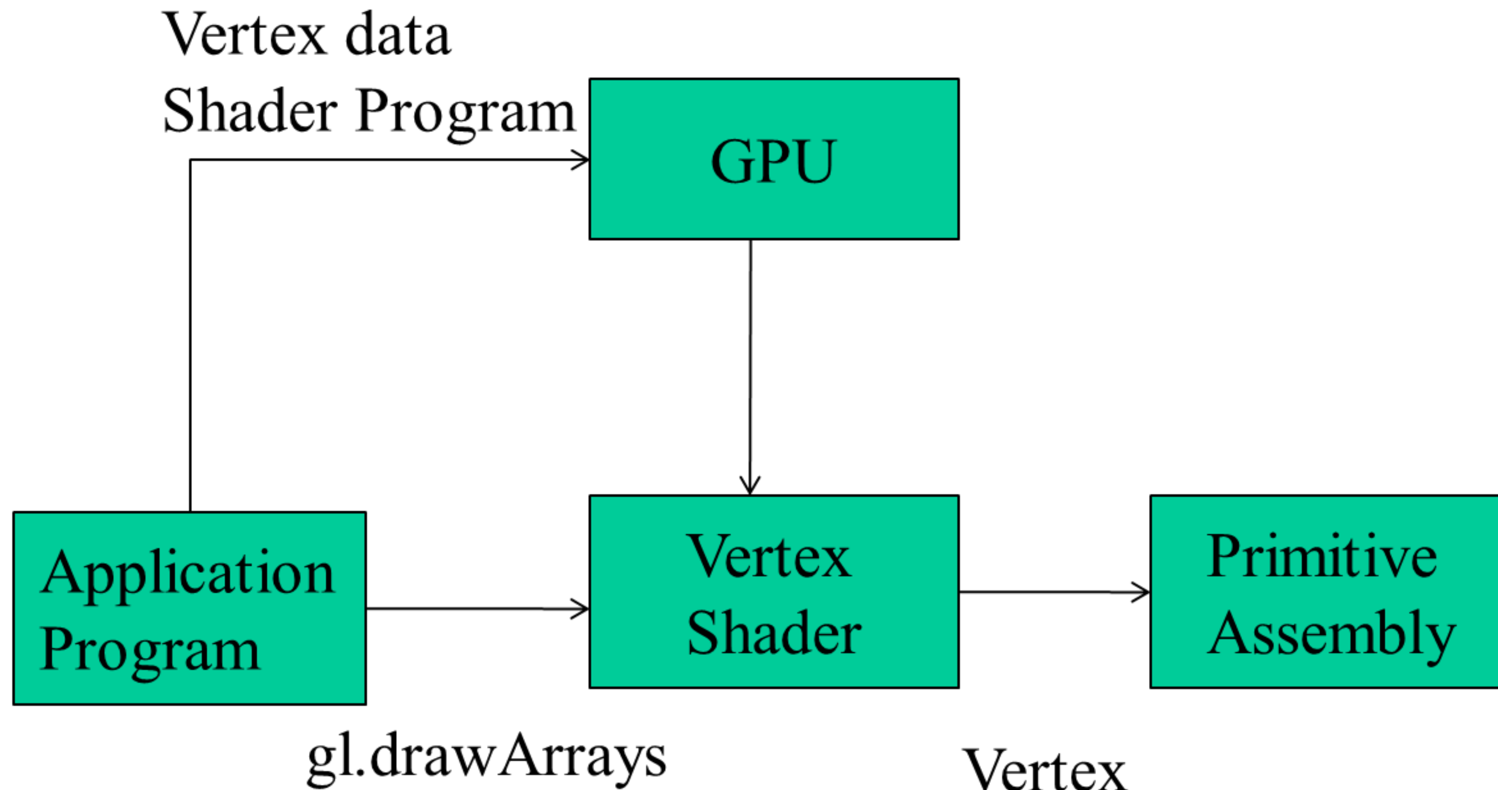
- Which primitive types?
  - gl.POINTS
  - gl.LINES, gl.LINE_STRIP, gl.LINE_LOOP
  - gl.TRIANGLES, gl.TRIANGLE_STRIP, gl.TRIANGLE_FAN

WebGL primitives



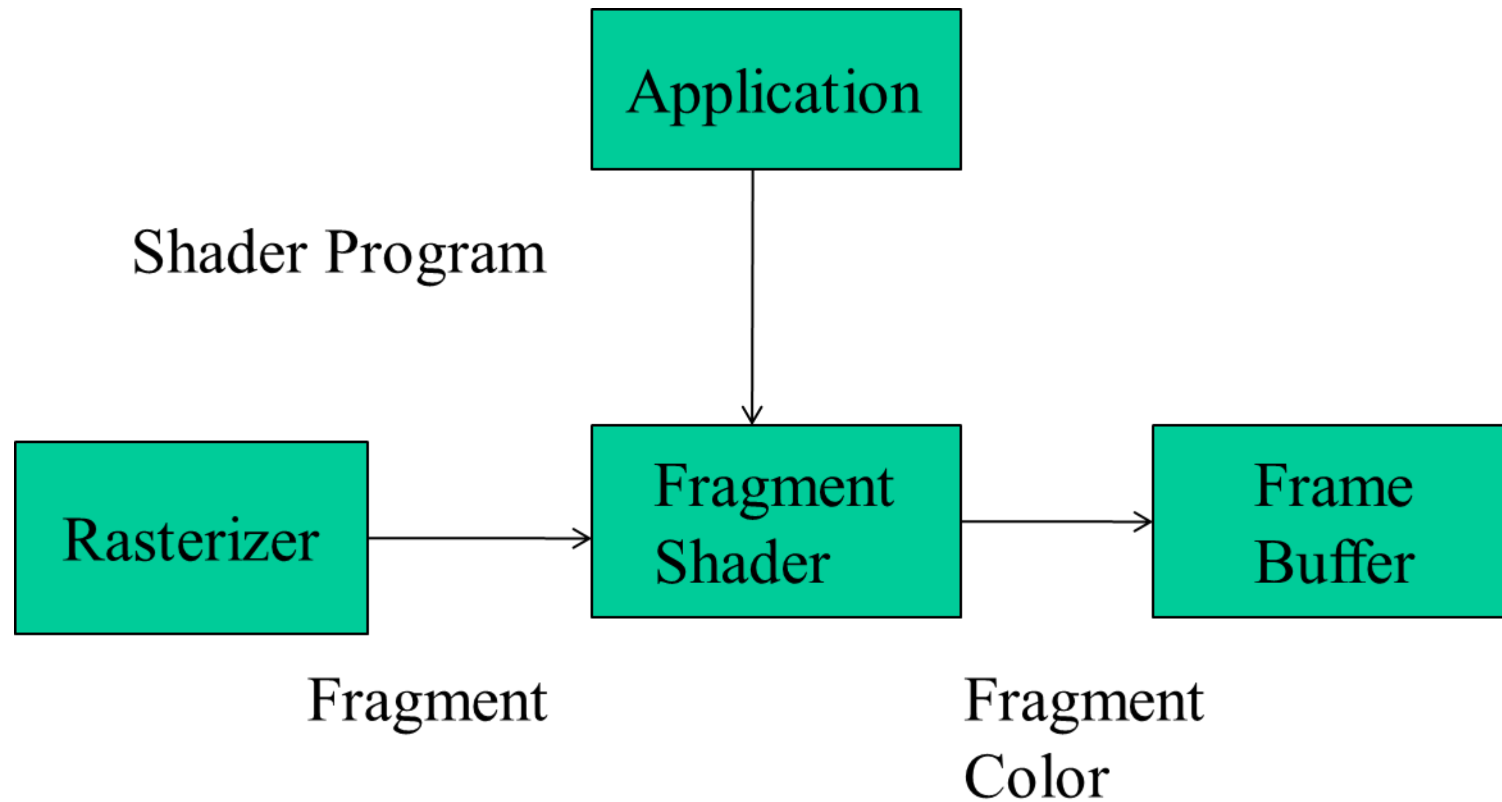
[Angel]

Execution model



[Angel]

Execution model



[Angel]

Linking shaders with application

- Read shaders
- Compile shaders
  - Check for errors !
- Create a program object
  - Container for shaders
- Link everything togeher
  - Check for errors !
- Link variables in application with variables in shaders
  - Vertex attributes
  - Uniform variables

WebGL programming

- Set up canvas to render onto
- Generate data in application
- Create shader programs
- Create buffer objects and load data into them
- “Connect” data locations with shader variables
- Render

WebGL – Application organization

- Do not put all code into a single HTML file !

- Put the setup in an HTML file
- And the application in a separate JavaScript file

WebGL – Application organization

- HTML file
  - contains shaders
  - brings in utilities and application JS file
  - describes page elements: buttons, menus
  - sets up canvas element

- JS file
  - initializes WebGL context
  - sets up VBOs
  - contains listeners for interaction
  - sets up required transformation matrices
  - reads, compiles and links shaders
  - triggers rendering