

# 3D Web App Creation



Fivos Doganis  
[fivos.doganis@gmail.com](mailto:fivos.doganis@gmail.com)



# Contents

-  [WebGL](#)
  - [Basic, Advanced](#)
  - [Optimizations, Shaders](#)
  - [Other APIs, WebAssembly](#)
-  [THREE.js](#)
  - [Theory](#)
  - [Setup](#)
  - [Basic, Editor, Advanced, Physics, 3D Models](#)
  - [Projects, Wrappers](#)

# Why use Web technologies for 3D?

- **mobile** experiences
- **open** technology stack
  - **cross-platform**, no installation, no app store
  - open source
  - **non-proprietary** (unlike Unity or Unreal Engine)
  - **free**
- leverage many other existing web APIs

# Steps

- Run
- Debug
- Build
- Modify
- Create

# Run: check your drivers

- Update your graphics card drivers if possible
- **Linux:** if your GPU is not properly detected, [uninstall NVIDIA drivers](#) and use the default ones provided by Ubuntu

```
sudo apt-get remove --purge '^nvidia-.*'  
sudo apt-get install ubuntu-desktop  
sudo rm /etc/X11/xorg.conf  
echo 'nouveau' | sudo tee -a /etc/modules  
reboot
```

# Run: testing on your mobile

- Check that your smartphone can read [QR codes](#)
- iOS : default Camera app
- Android
  - use Google Chrome + scan button
  - or install a **trustworthy** QR code scanning app like [Trend Micro](#)
- Other 100% web based alternatives
  - [webqr.com](http://webqr.com)
  - [qrcodescan.in](http://qrcodescan.in)

# Debug

- There will be bugs :) 
- F12
- debugger statement
- Tips:
  - <https://webglfundamentals.org/webgl/lessons/webgl-setup-and-installation.html>
  - <https://threejs.org/manual/en/debugging-javascript.html>
  - <https://www.khronos.org/webgl/wiki/Debugging>

# Build: tools

- Web development
  - browser (Firefox, Chrome, Safari Mobile)
  - Git (optional yet extremely useful)
  - editor (VSCode), or [Glitch](#) (slow, but no installation needed!)
- Technologies: HTML, JS, CSS, WebGL, THREE.js
- Optional:
  - OpenStreetMap, OpenLayers, CesiumJS, [Géoservices IGN](#)

# Browser installation

- Firefox installed by default
  - should be enough!
- Chrome
  - to test compatibility as well as some special features
  - install latest version (97+) on mobile
  - you may install Chromium on your desktop
    - open-source without proprietary services

```
sudo apt-get install chromium-browser
```

# Git installation

```
sudo apt-get install git  
  
git config --global user.name "myusername"  
git config --global user.email myname@mymailprovider.com
```

# VSCode installation

```
sudo apt update  
sudo apt install software-properties-common apt-transport-https wget  
  
wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | sudo apt-key add -  
  
sudo add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main"  
  
sudo apt install code
```

## Remove GPG warnings

```
sudo gpgconf --kill dirmngr  
sudo chown -R $USER:$USER ~/.gnupg
```

# Customize VS Code

- Avoid UI blinking by changing the settings:

```
set window.titleBarStyle to custom
```

- Recommended extensions
  - [Live Server](#)
  - [Git Graph](#) and/or [Git Lens](#)
  - [glTF Model Viewer, glTF Tools](#)
  - [WebGL GLSL Editor, glsl-canvas](#)
  - [Todo Tree, Color Highlight](#)



Settings



format

## File &gt; Settings &gt; Format on Save ★

83 Settings Found



User Settings   Workspace Settings



Commonly Used (1)

- ▲ Text Editor (8)
  - Cursor (2)
  - Formatting (4)
- ▲ Workbench (1)
  - Editor Managem... (1)
- ▲ Features (2)
  - Terminal (2)
- ▲ Extensions (71)



CSS (9)

Emmet (1)

HTML (12)

JSON (2)

LESS (6)

Npm (1)

SCSS (Sass) (7)

TypeScript (33)

**Editor: Format On Paste**

Controls whether the editor should automatically format the pasted content. A formatter must be available and the formatter should be able to format a range in a document.

**Editor: Format On Save**

Format a file on save. A formatter must be available, the file must not be saved after delay, and the editor must not be shutting down.

**Editor: Format On Save Timeout**

Timeout in milliseconds after which the formatting that is run on file save is cancelled.

750

**Editor: Format On Type**

Controls whether the editor should automatically format the line after typing.

**Editor > Parameter Hints: Enabled**

Enables a pop-up that shows parameter documentation and type information as you type.

# Development environment setup

- Files cannot be loaded from disk without a user action
- **CORS: Cross Origin Resource Sharing**
  - one of the many web browser security measures

→ need to run a server like Live Server, or:

```
$ cd /home/somedir  
$ python -m SimpleHTTPServer
```

And then point your browser at `http://localhost:8000`

# Glitch: online interactive editor

- <https://glitch.com/>
- [THREE.js example](#)

But also [VSCode.dev](https://VSCode.dev), Codepen.io, Repl.it etc.

# WebGL Exercises

inspired by

- [webglacademy](#)
- [webglfundamentals](#) ★
- [webgl2fundamentals](#)

# Check that WebGL is supported

- <https://caniuse.com/webgl>
- <https://webglreport.com/>

# Reminder: Minimalistic web page

```
<!DOCTYPE html>
<html>

<head>
    <meta charset='utf-8'>
</head>

<body>
</body>

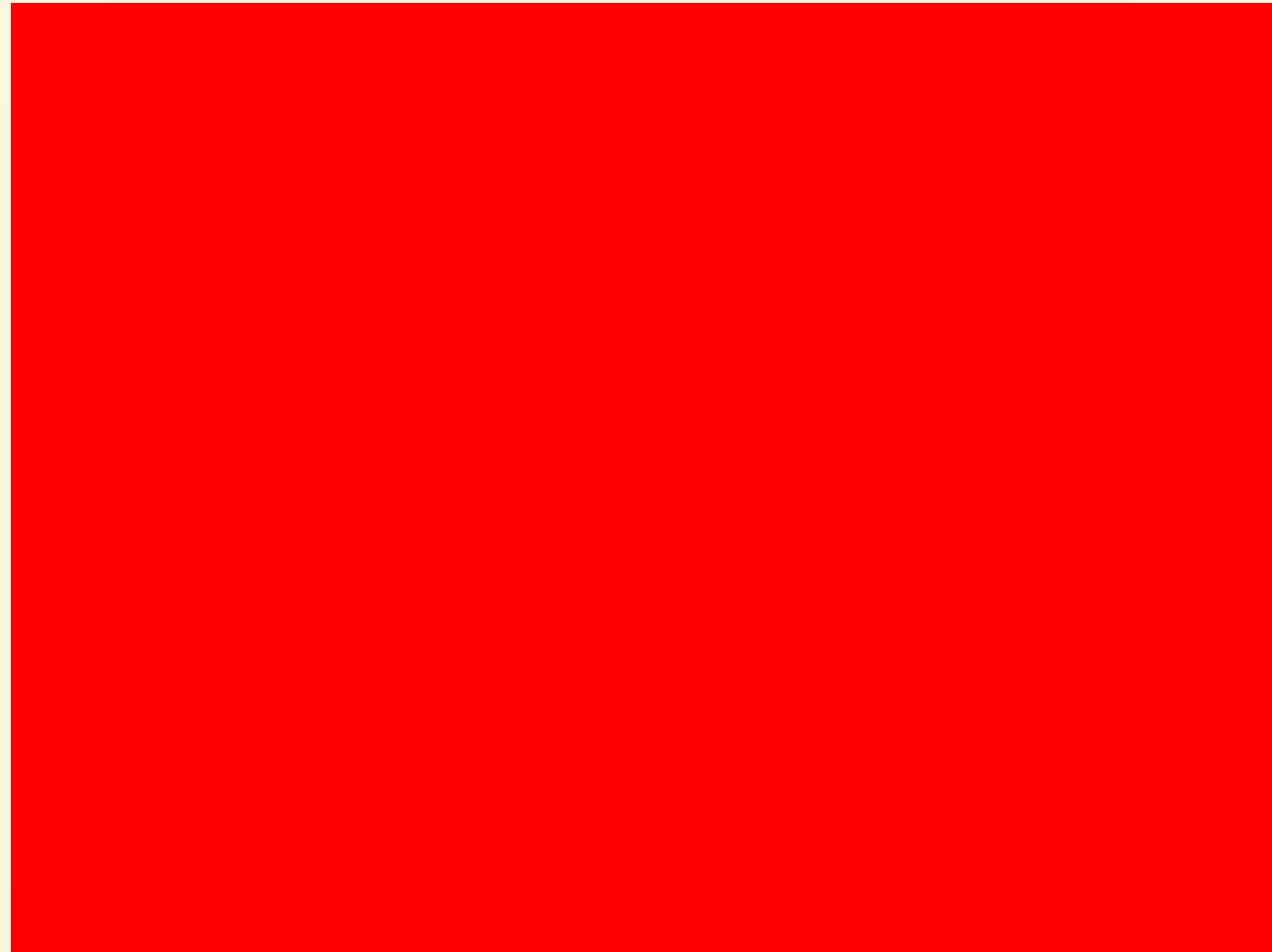
</html>
```

# Reminder: Minimalistic drawing program using the Canvas API

```
<!DOCTYPE html>
<html>

<head>
    <meta charset='utf-8'>
</head>
<body>
    <canvas width='640' height='480'></canvas>
    <script>
        const canvas = document.querySelector('canvas');
        const ctx = canvas.getContext('2d'); // get 2D rendering context, on which we'll draw
        ctx.fillStyle = 'rgba(255, 0, 0, 1)'; // a.k.a. 'red': opaque red
        ctx.fillRect(0, 0, canvas.width, canvas.height); // uses current color
    </script>
</body>

</html>
```



# "There Is Also Canvas"

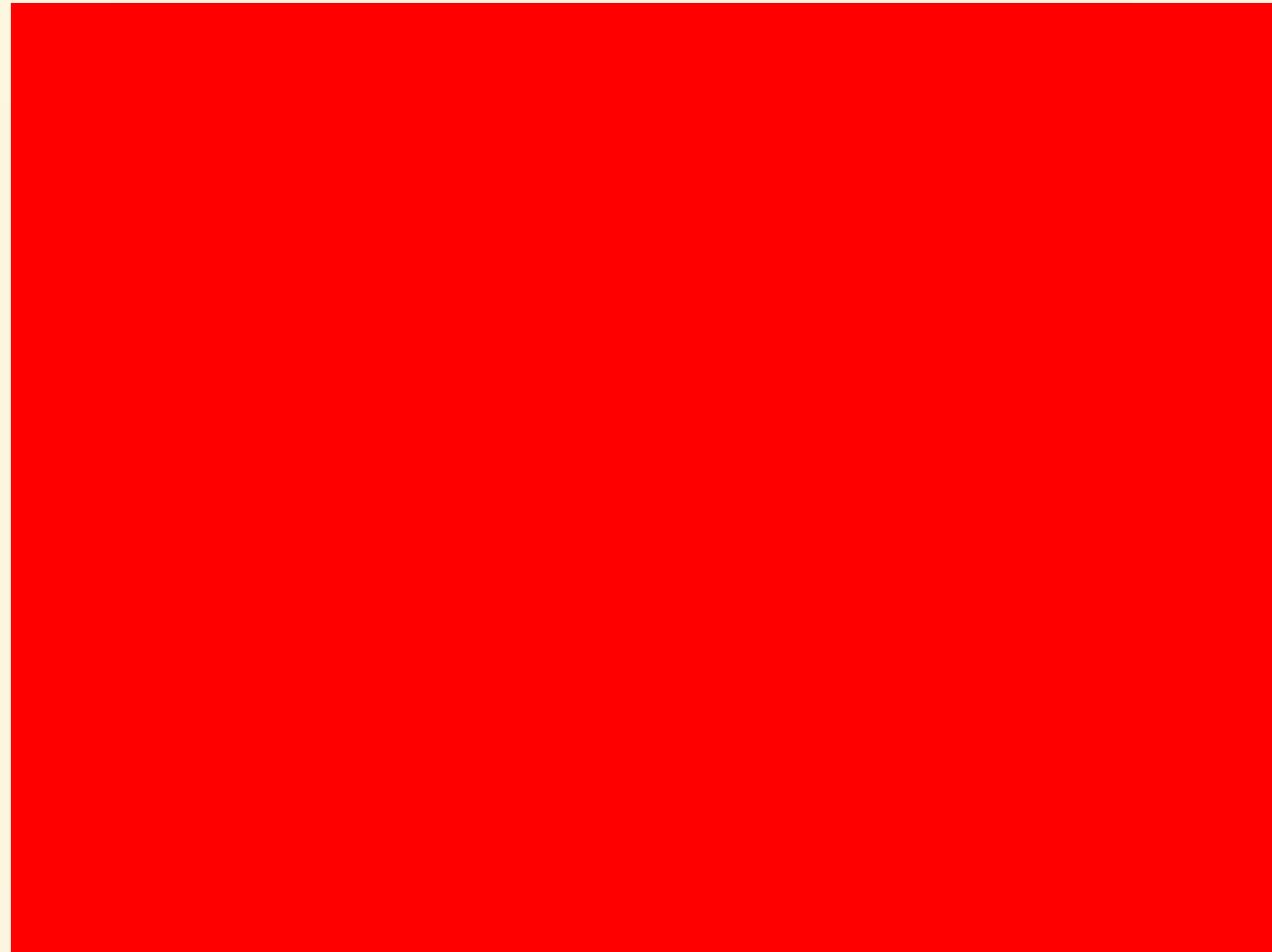
Bruno Imbrizi



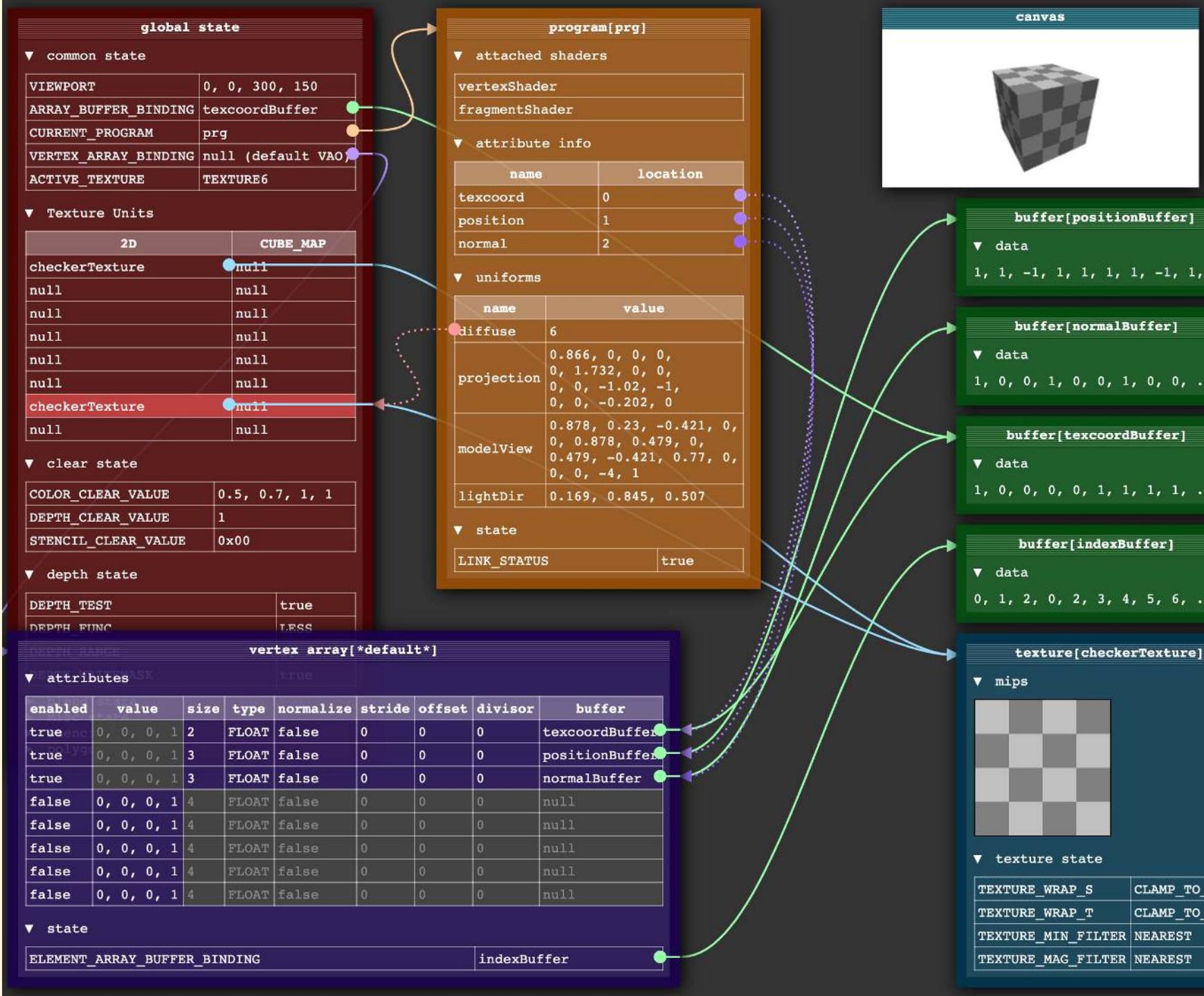
# Minimalistic WebGL program: to ensure that everything works fine

⚠️ **no error checks**, for clarity reasons (*don't do this at home!* 😊)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset='utf-8'>
</head>
<body>
  <canvas width='640' height='480'></canvas>
  <script>
    const canvas = document.querySelector('canvas');
    /** @type {WebGLRenderingContext} */
    const gl = canvas.getContext('webgl'); // instead of '2d'
    gl.clearColor(1., 0., 0., 1.); // RGBA: opaque red
    gl.clear(gl.COLOR_BUFFER_BIT); // uses current color (state machine)
  </script>
</body>
</html>
```



# States



## Minimalistic "useful" program: shaders, but no vertex buffer yet

Add this code after `gl.clear`:

```
// vertex shader
const vs_source = `

void main() {
    gl_Position = vec4(0., 0., 0., 1.); // center
    gl_PointSize = 120.0;
}`;

// fragment shader
const fs_source = `

precision mediump float;

void main() {
    gl_FragColor = vec4(0., 1., 0., 1.); // green
}`;
```

## Additional code: useful functions (still no error checks!)

```
function buildShader(gl, shaderSource, shaderType) {  
    const shader = gl.createShader(shaderType); // Create the shader object  
    gl.shaderSource(shader, shaderSource); // Load the shader source  
    gl.compileShader(shader); // Compile the shader  
    return shader;  
}  
  
function createProgram(gl, shaders) {  
    const program = gl.createProgram();  
    shaders.forEach(function(shader) {  
        gl.attachShader(program, shader);  
    });  
    gl.linkProgram(program);  
    return program;  
}
```

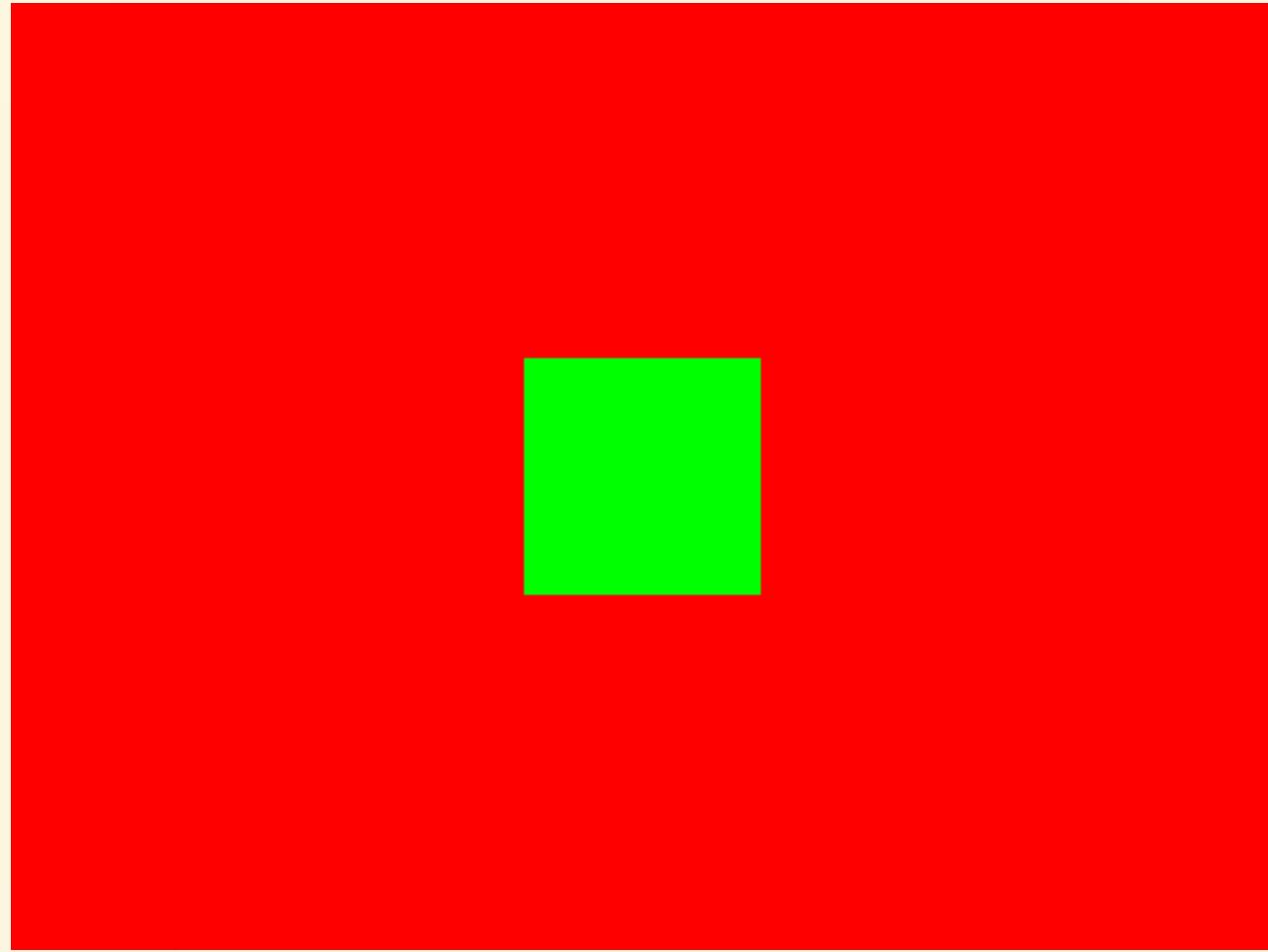
## Finally : load shaders, program and render

```
// load and compile the shaders
const vs = buildShader(gl, vs_source, gl.VERTEX_SHADER);
const fs = buildShader(gl, fs_source, gl.FRAGMENT_SHADER);

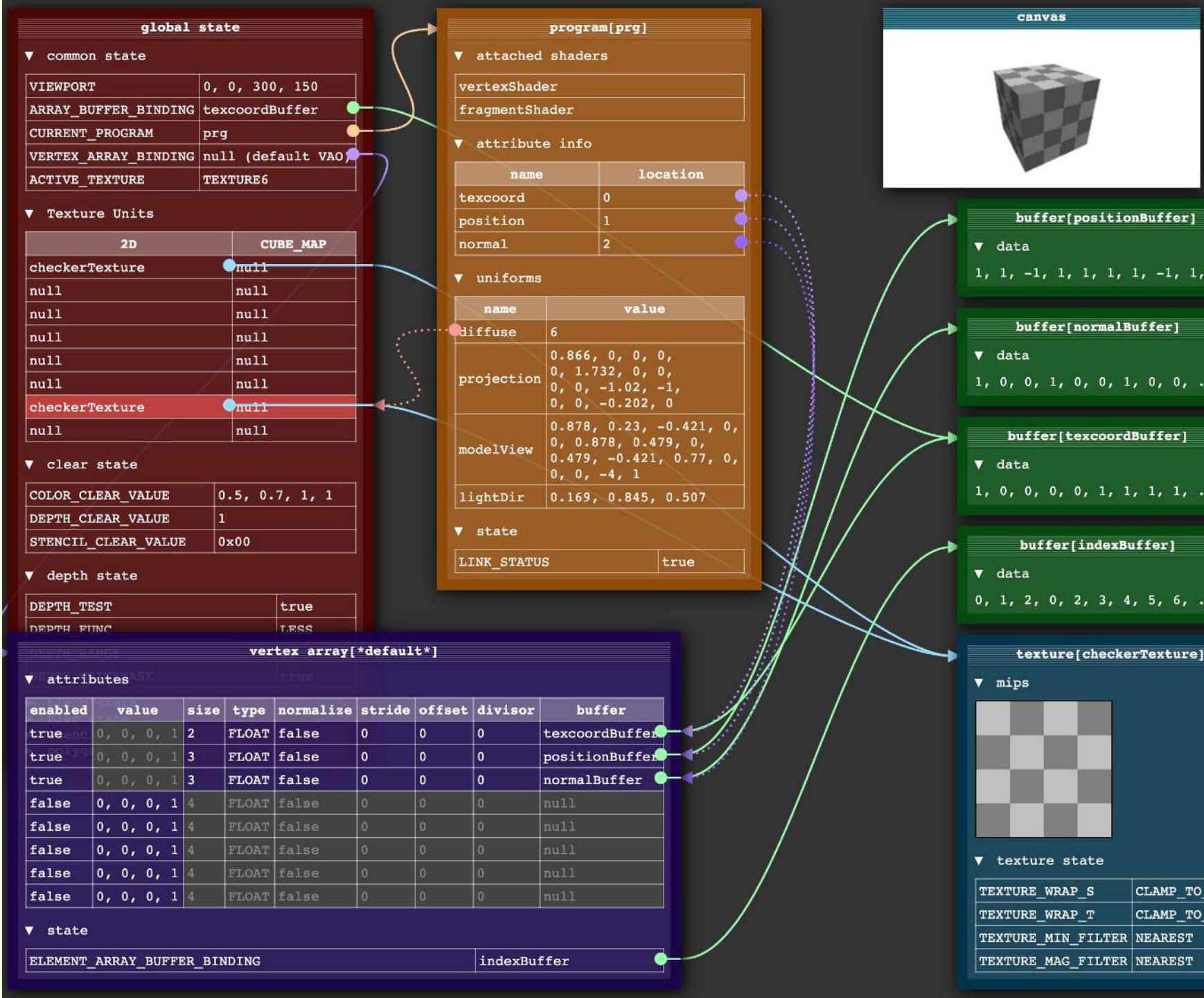
// Create program on the GPU!
const program = createProgram(gl, [vs, fs]);

// Set current program (WebGL is a state machine!)
gl.useProgram(program);

// Draw 1 big point, see shaders
const offset = 0;
const count = 1;
gl.drawArrays(gl.POINTS, offset, count);
```



# States

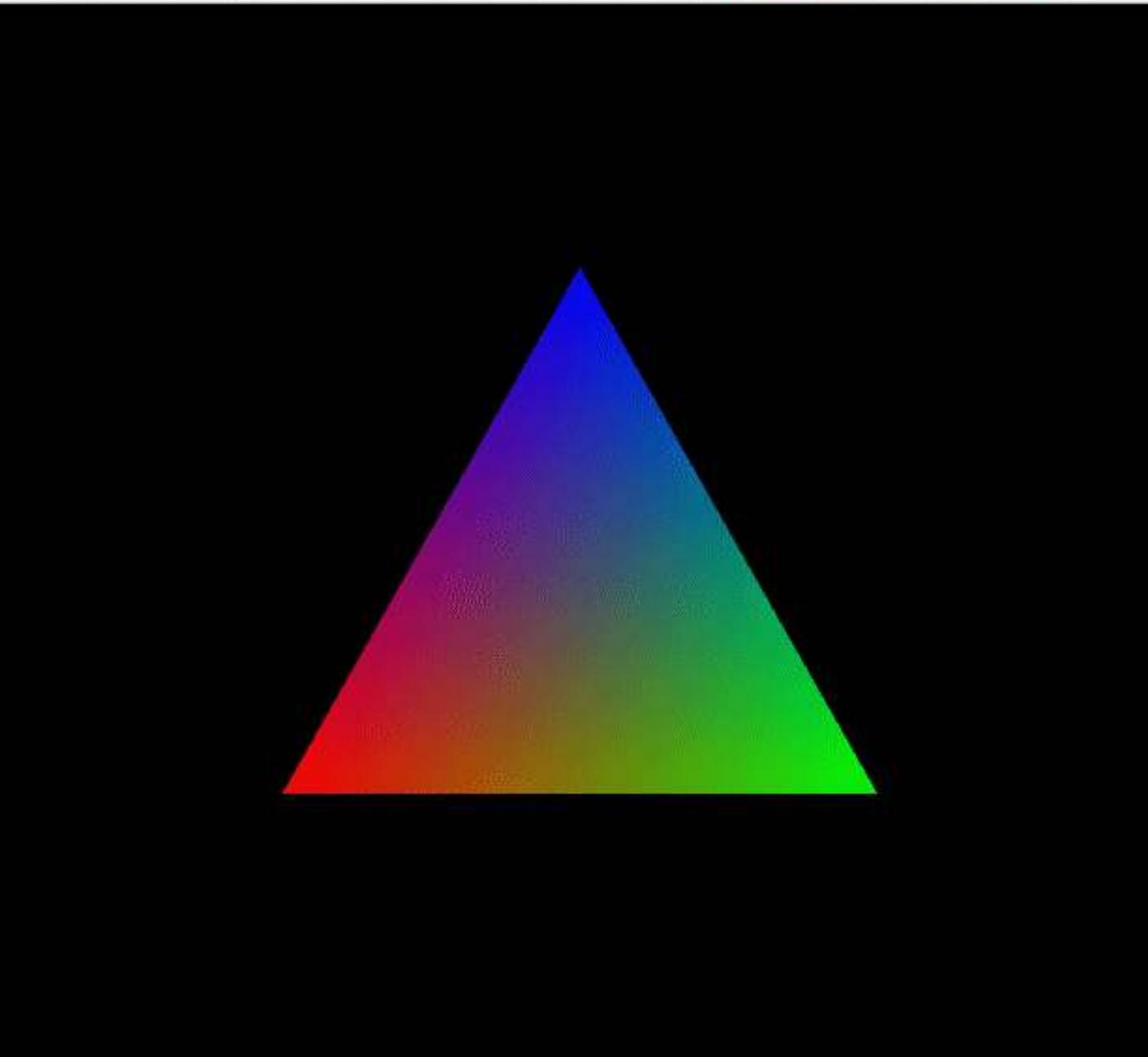


# I KNOW

# WEBGL



**SHOW ME**



# My first triangle

see webglfundamentals

# Initialization

# 1 . Data

- Describe triangle geometry with `Float32Array` in clip space

```
const vertices = new Float32Array(  
    0.5, 0.5,  
    -0.5, 0.5,  
    -0.5, -0.5); // 2D points: 3 * (x, y) coordinates
```

- Create Vertex Buffer Object (VBO) + **UPLOAD** to the GPU

```
const vbo = gl.createBuffer(); // create Vertex Buffer Object (VBO) id  
// Set current VBO: bind 'vbo' to the ARRAY_BUFFER bind point,  
// a global variable internal to WebGL (state machine!)  
gl.bindBuffer(gl.ARRAY_BUFFER, vbo);  
// UPLOAD current VBO to GPU, where it will be processed by the shaders  
// NOTE: STATIC_DRAW: optimization hint for WebGL: our data won't change  
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
```

## 2 . Shaders

which will use buffer data

- Vertex Shader: `gl_Position` will be used by the fragment shader

```
attribute vec2 a_position; // IN, from buffer: 2D point
void main() {
    gl_Position = vec4(a_position, 0.0, 1.0); // a_position.x, a_position.y, 0, 1, used by the fragment shader
}
```

- Fragment Shader: `gl_FragColor` is the final fragment color

```
precision mediump float; // float accuracy: lowp, mediump, highp
uniform vec4 u_color; // UNIFORM == CONSTANT for entire shader program

void main() {
    gl_FragColor = u_color; // final framebuffer color: RGBA
}
```

# Detecting shader compilation errors

Call this before `gl.useProgram` (optional, slow, but often useful 😊)

```
function checkShaders(gl, vs, fs, program) {  
    if (!gl.getShaderParameter(vs, gl.COMPILE_STATUS))  
        console.error(gl.getShaderInfoLog(vs));  
  
    if (!gl.getShaderParameter(fs, gl.COMPILE_STATUS))  
        console.error(gl.getShaderInfoLog(fs));  
  
    if (!gl.getProgramParameter(program, gl.LINK_STATUS))  
        console.error(gl.getProgramInfoLog(program));  
}
```

# **3 . Connecting Buffer and Shaders**

- Retrieve the variables declared in the shaders
  - costly call (like most `gl.getXXX` calls)
  - only do this during initialization
  - will be used during rendering
- ! use the exact same name that has been defined in the shader
  - arbitrary name, chosen by the developer! (`u_` for `uniform`, `a_` for `attribute`, `v_` for `varying` are common conventions)

```
// Retrieve 'u_color' shader UNIFORM variable as an id
const u_colorLoc = gl.getUniformLocation(program, 'u_color');
```

```
// Retrieve 'a_position' shader ATTRIBUTE variable as an id
const a_positionLoc = gl.getAttributeLocation(program, 'a_position');
```

# 4 . Rendering!

Buffers and Shaders are ready, we still need to:

- define states
- describe buffer layout (often complex since it is very flexible!)
- draw the scene

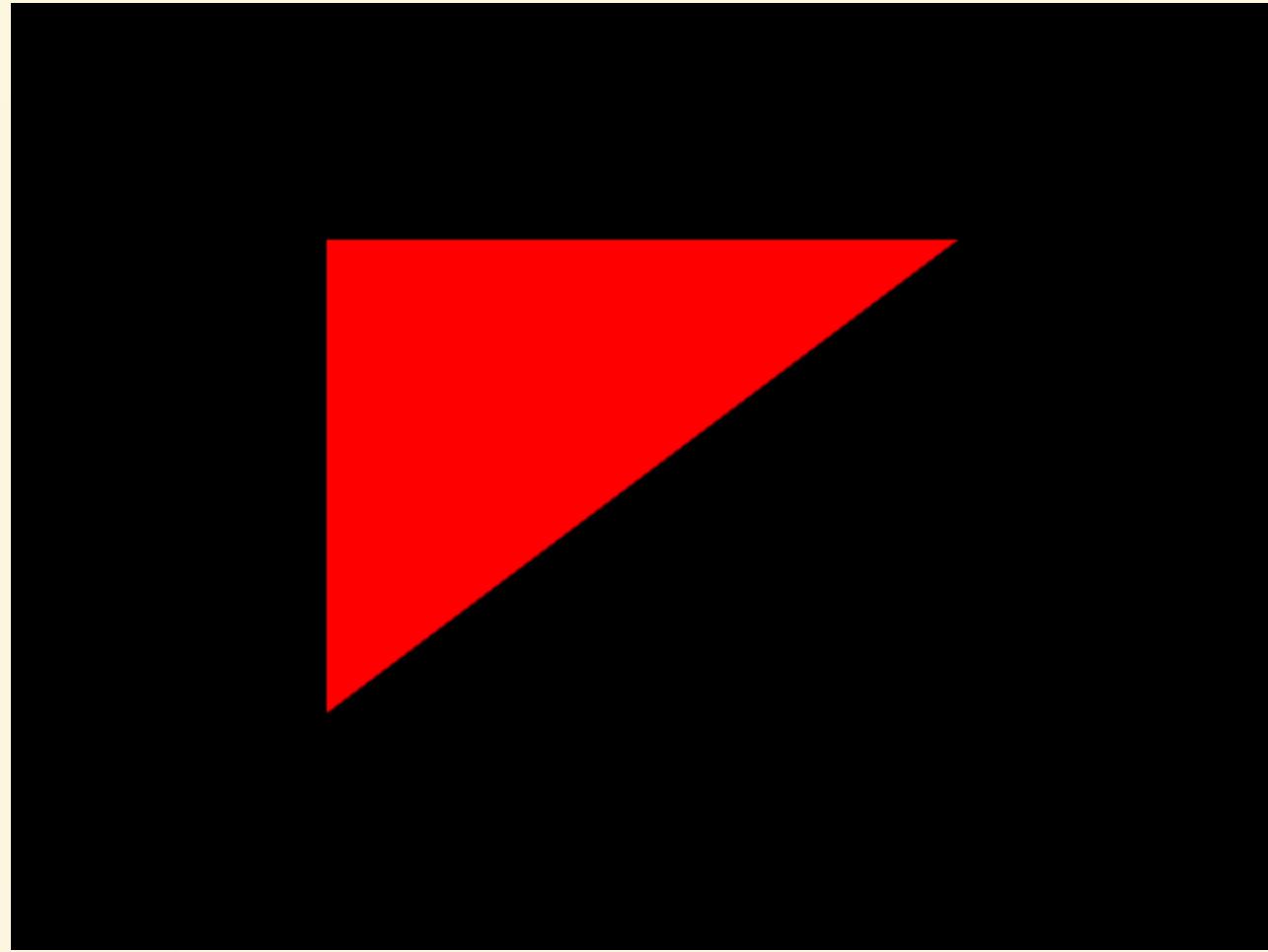
```
gl.clearColor(0., 0., 0., 1.); // Set current clear color (black)
gl.clear(gl.COLOR_BUFFER_BIT); // Clear the canvas to current color

gl.useProgram(program); // Set current program (pair of shaders)

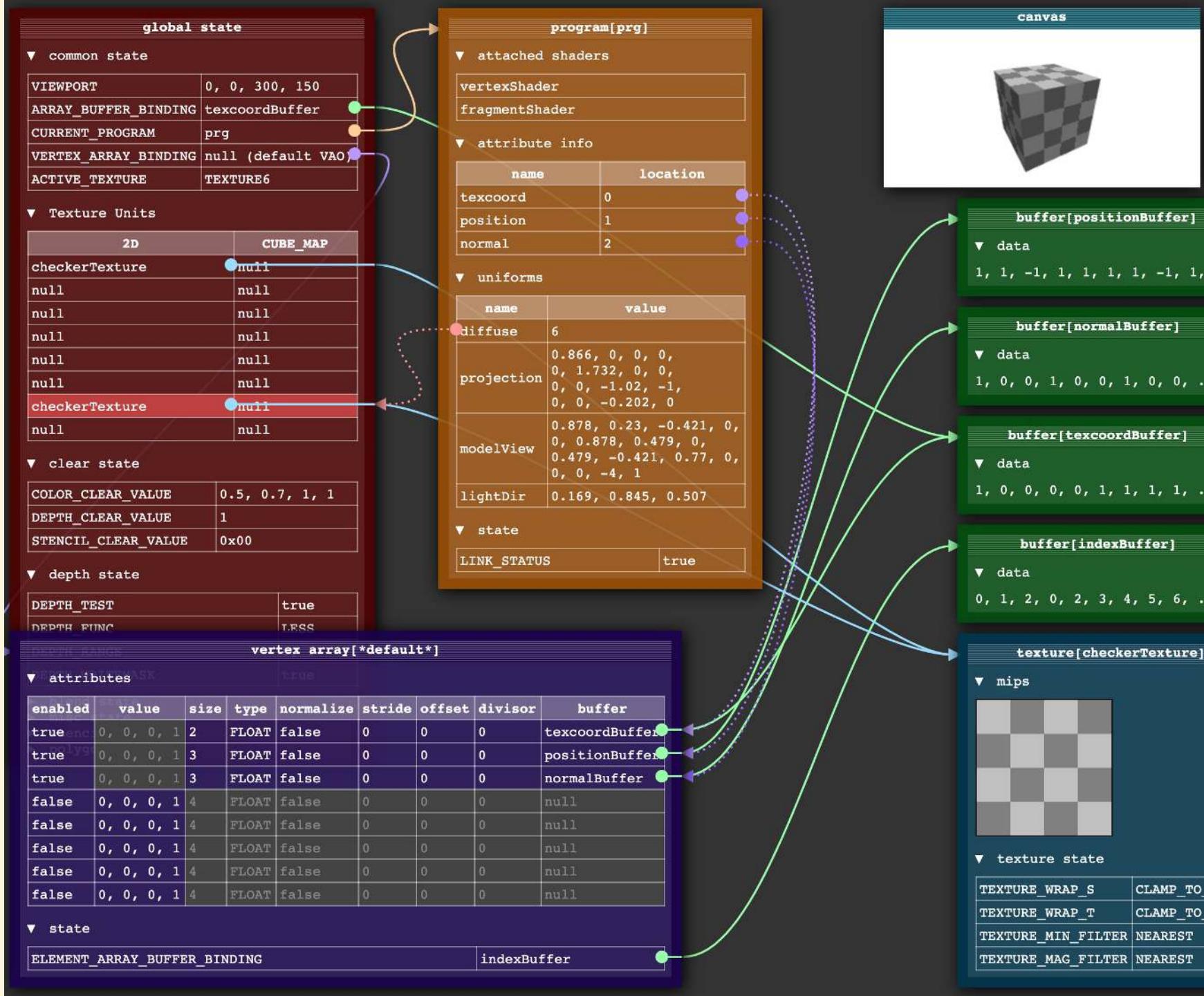
// Set the color (constant triangle color, see shader)
gl.uniform4fv(u_colorLoc, [1.0, 0.0, 0.0, 1.0]);
// Tell WebGL how to take data from the VBO
// and supply it to the attribute in the shader.
gl.enableVertexAttribArray(a_positionLoc); // Turn the attribute on
gl.bindBuffer(gl.ARRAY_BUFFER, vbo); // set current vbo
// Tell the attribute how to get data out of positionBuffer
// (ARRAY_BUFFER)
const size = 2; // 2 components per iteration
const type = gl.FLOAT; // the data is 32bit floats
const normalize = false; // don't normalize the data
// stride: 0 = move forward size * sizeof(type) each iteration
// to get the next position
const stride = 0;
const offset = 0; // start at the beginning of the buffer
gl.vertexAttribPointer(a_positionLoc, size, type, normalize, stride, offset);
```

- Draw: here we'll be using `gl.drawArrays`

```
// primitiveType == gl.TRIANGLES:  
// each time our vertex shader is run 3 times,  
// WebGL will draw a triangle  
// based on the 3 values we set gl_Position to (see shader)  
const primitiveType = gl.TRIANGLES;  
  
// Start index of the first vertex  
// Must be a valid multiple of the size of the given type.  
const startIndex = 0;  
  
// Execute our vertex shader 3 times,  
// using 2 elements from the array (see size 2 above)  
// setting a_position.x and a_position.y  
const count = 3;  
gl.drawArrays(primitiveType, startIndex, count);
```



# States



# Almost there!

We still need to define one color per vertex instead of the constant  
uniform color  
cf. [webglfundamentals](#)

# 1 . Data

adding a color attribute per vertex, with a new array and a new buffer!

```

// 2D points: 3 * (x, y) coordinates
// ( $\sqrt{3} / 2 - 0.5$ ) * 640 / 480 == 0.4880338..
const vertices = new Float32Array([-0.75, -0.5,
                                    0.75, -0.5,
                                    0., 0.49]);
// create Vertex Buffer Object (VBO) id
const vertexBuffer = gl.createBuffer();

// set current VBO (WebGL is a state machine!)
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);

// UPLOAD vertexBuffer VBO to GPU,
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

// ** NEW **

// RGB colors: 3 * (r, g, b, a) values
const colors = new Float32Array([ 1., 0., 0., 1.,
                                 0., 1., 0., 1.,
                                 0., 0., 1., 1.]);
// create Vertex Buffer Object (VBO) id
const colorBuffer = gl.createBuffer();

// set current VBO (WebGL is a state machine!)
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);

// UPLOAD colorBuffer VBO to GPU,
gl.bufferData(gl.ARRAY_BUFFER, colors, gl.STATIC_DRAW);

```

## 2 . Shaders

we use the new color attribute that we choose to name `a_color`  
(we could name it `mylittlevertexcolor` but it looks less professional)

- Vertex Shader: new **varying** variable, will be **interpolated**

```
attribute vec2 a_position; // IN, from buffer: 2D point
attribute vec4 a_color; // IN, from buffer: RGB color

varying vec4 v_color; // OUT, to fragment shader

void main() {
    v_color = a_color; // color passthrough, sent to fragment shader (interpolated)

    gl_Position = vec4(a_position, 0.0, 1.0); // used by the fragment shader
}
```

- Fragment Shader

```
precision mediump float; // float accuracy: lowp, mediump, highp

varying vec4 v_color; // IN, INTERPOLATED color from vertex shader

void main() {
    gl_FragColor = vec4(v_color); // final framebuffer color: RGBA
}
```

# 3 . Connecting Buffer and Shaders

- Retrieve the variables declared in the shaders
  - costly call (like most `gl.getXXX` calls)
  - only do this during initialization
  - will be used during rendering
- ! use the exact same name that has been defined in the shader

```
// Retrieve 'a_color' shader ATTRIBUTE variable as an id
const a_colorLoc = gl.getAttributeLocation(program, 'a_color');

// Retrieve 'a_position' shader ATTRIBUTE variable as an id
const a_positionLoc = gl.getAttributeLocation(program, 'a_position');
```

# 4 . Rendering!

Buffers and Shaders are ready, we still need to:

- define states
- describe buffer layout (often complex since it is very flexible!)
- draw the scene

```
gl.clearColor(0., 0., 0., 1.);  
gl.clear(gl.COLOR_BUFFER_BIT);  
  
gl.useProgram(program); // Set current program  
  
{  
    // Turn the attribute on  
    gl.enableVertexAttribArray(a_positionLoc);  
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer); // set current vbo  
    // Tell the attribute how to get data out of positionBuffer  
    // (ARRAY_BUFFER)  
    const size = 2; // 2 components per iteration  
    const type = gl.FLOAT; // the data is 32bit floats  
    const normalize = false; // don't normalize the data  
    // stride: 0 = move forward size * sizeof(type) each iteration  
    // to get the next position  
    const stride = 0;  
    const offset = 0; // start at the beginning of the buffer  
    gl.vertexAttribPointer(a_positionLoc, size, type, normalize, stride, offset);  
}
```

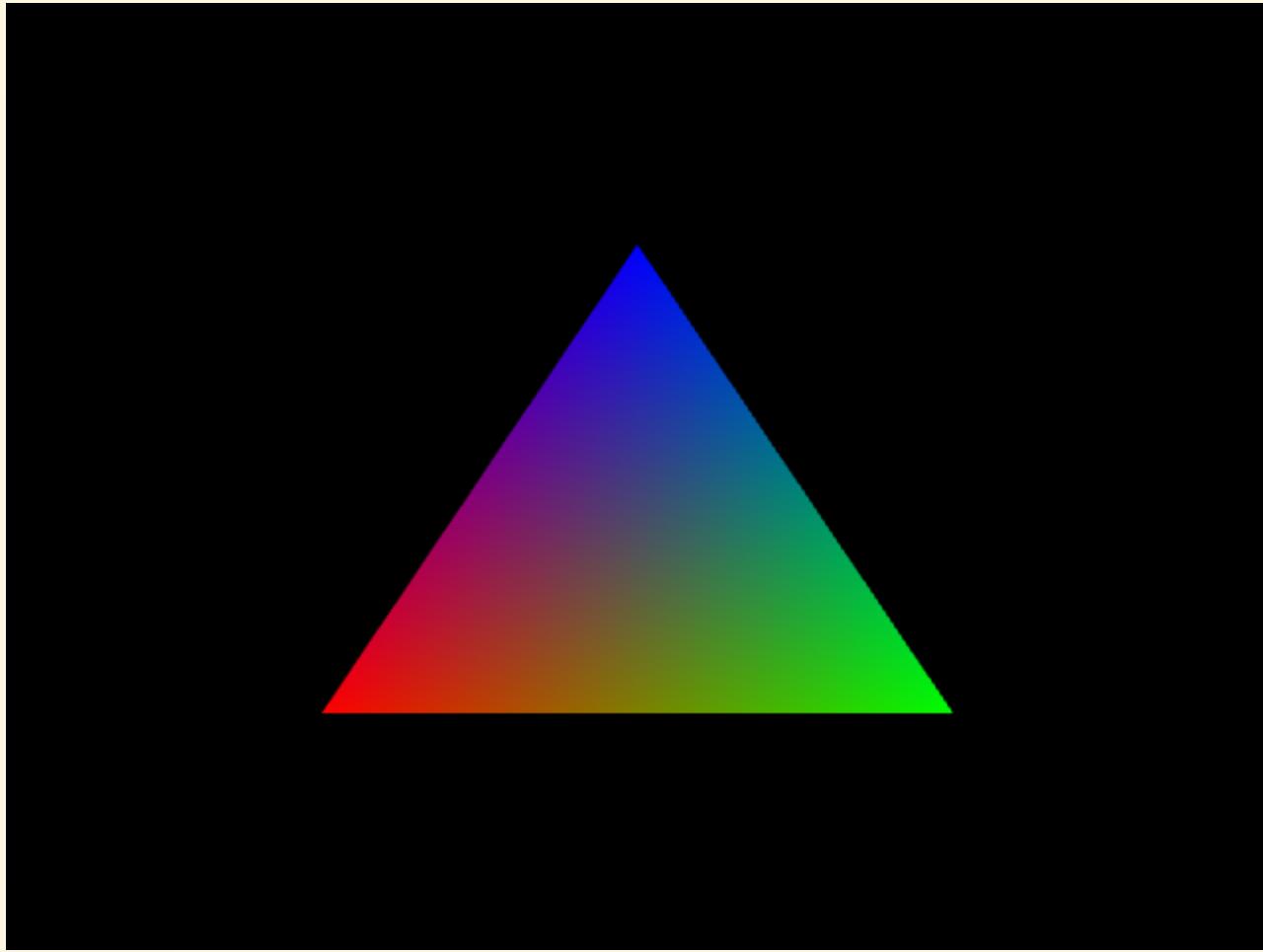
- Same steps for `colorBuffer`

**NOTE:** Refactoring! In real life we try and avoid repetition: [DRY](#)

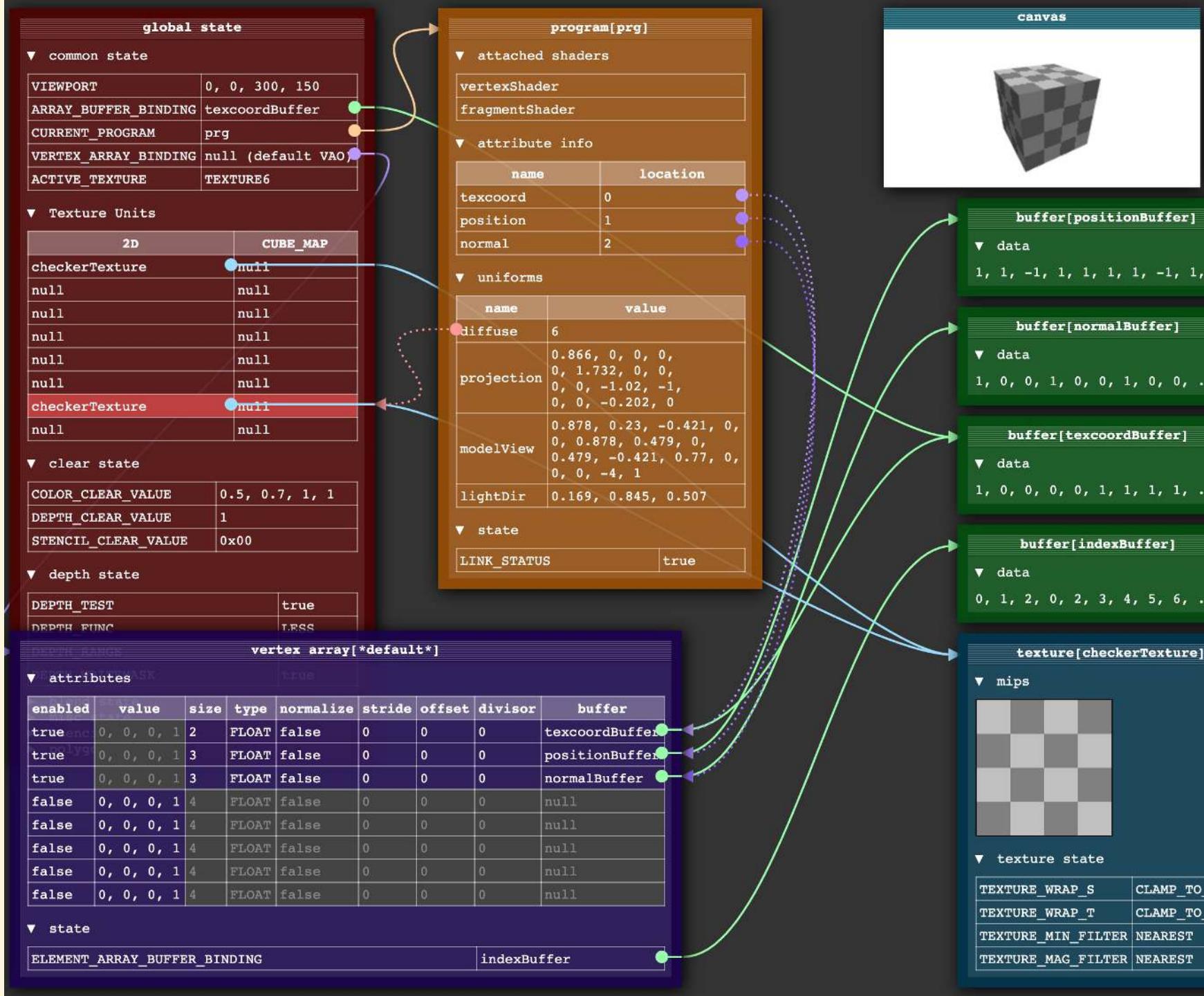
```
{  
    // Tell WebGL how to take data from the VBO  
    // and supply it to the attribute in the shader.  
    gl.enableVertexAttribArray(a_colorLoc); // Turn the attribute on  
    gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);  
    // Tell the color attribute how to get data out of colorBuffer (ARRAY_BUFFER)  
    const size = 4; // NOTE: 4 components per iteration  
    const type = gl.FLOAT; // the data is 32bit floats  
    const normalize = false; // don't normalize the data  
    // stride: 0 = move forward size * sizeof(type) each iteration  
    // to get the next position  
    const stride = 0;  
    const offset = 0; // start at the beginning of the buffer  
    gl.vertexAttribPointer( a_colorLoc, size, type, normalize, stride, offset );  
}
```

- Draw: same code as before!

```
// primitiveType == gl.TRIANGLES:  
// each time our vertex shader is run 3 times,  
// WebGL will draw a triangle  
// based on the 3 values we set gl_Position to (see shader)  
const primitiveType = gl.TRIANGLES;  
  
// Start index of the first vertex  
// Must be a valid multiple of the size of the given type.  
const startIndex = 0;  
  
// Execute our vertex shader 3 times,  
// using 2 elements from the array (see size 2 above)  
// setting a_position.x and a_position.y  
const count = 3;  
gl.drawArrays(primitiveType, startIndex, count);
```



# States





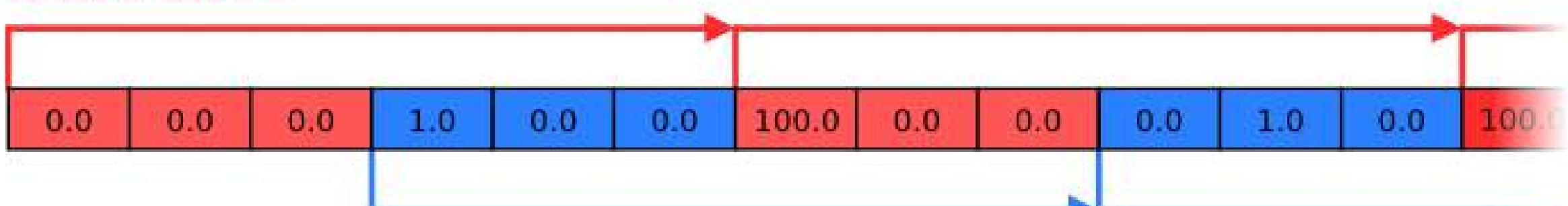
# Variations

- A single buffer, interlaced and indexed + `gl.DrawElements`
  - indexing: share a vertex between triangles (no need to repeat)
  - is it really an optimization? (platform specific profiling needed)
  - brief history of vertex specifications buffer types and layout, vocabulary, structures, illustrations
- WebGL 2 version

# Interlaced buffer

```
vertex stride = sizeof(vertex)
              = 6 * sizeof(float)
              = 6 * 4
              = 24
```

vertex offset = 0



```
colour offset = sizeof(vector3)
               = 3 * sizeof(float)
               = 3 * 4
               = 12
```

```
colour stride = sizeof(vertex)
                 = 6 * sizeof(float)
                 = 6 * 4
                 = 24
```

```
// POINTS : position + color
const vertexBuffer = new Float32Array([
-0.5, -0.5, // vertex 1 : left
1., 0., 0., // color 1 : red
0.5, -0.5, // vertex 2 : right
0., 1., 0., // color 2 : green
0., 0.48, // vertex 3 : top
0., 0., 1. // color 3 : blue
]);

const vbo = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vbo);
gl.bufferData(gl.ARRAY_BUFFER, vertexBuffer, gl.STATIC_DRAW); // send to GPU

// FACES : indices
const indices = new Uint16Array([0, 1, 2]);
var indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW); // send to GPU
```

- Rendering: `gl.DrawElements` + `window.requestAnimationFrame`

```
gl.enableVertexAttribArray(a_colorLoc);
gl.enableVertexAttribArray(a_positionLoc);
gl.useProgram(program);
gl.clearColor(0.0, 0.0, 0.0, 1.0);
const animate = () => {
    gl.viewport(0.0, 0.0, canvas.width, canvas.height);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.bindBuffer(gl.ARRAY_BUFFER, vbo);

    gl.vertexAttribPointer(a_positionLoc, 2, gl.FLOAT, false, 4*(2+3), 0) ;
    gl.vertexAttribPointer(a_colorLoc, 3, gl.FLOAT, false, 4*(2+3), 2*4) ;

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
    gl.drawElements(gl.TRIANGLES, 3, gl.UNSIGNED_SHORT, 0);
    gl.flush();
    window.requestAnimationFrame(animate);
};

animate();
```

# More examples

- [spinning triangle](#)
  -  using **matrices**
- [spinning cube](#)
  -  advanced buffer layout
- [textures](#)

# WebGL Textured Cube (HTML)

```
<canvas id="canvas"></canvas>
<!-- vertex shader -->
<script id="vertex-shader-3d" type="x-shader/x-vertex">
attribute vec4 a_position;
attribute vec2 a_texcoord;

uniform mat4 u_matrix;

varying vec2 v_texcoord;

void main() {
    // Multiply the position by the matrix.
    gl_Position = u_matrix * a_position;

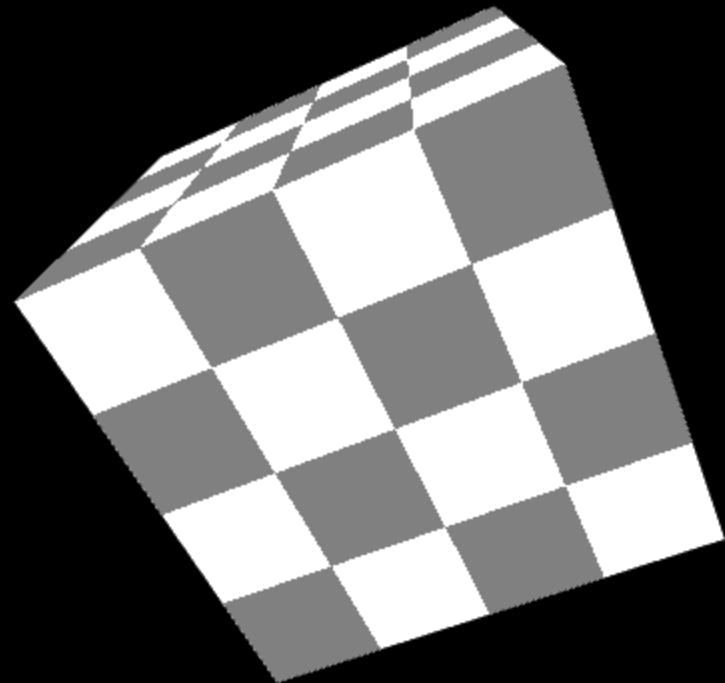
    // Pass the texcoord to the fragment shader.
    v_texcoord = a_texcoord;
}
</script>
<!-- fragment shader -->
<script id="fragment-shader-3d" type="x-shader/x-fragment">
precision medium float;

// Passed in from the vertex shader.
varying vec2 v_texcoord;

// The texture.
uniform sampler2D u_texture;

void main() {
    gl_FragColor = texture2D(u_texture, v_texcoord);
}
</script><!--
for most samples webgl-utils only provides shader compiling/linking and
canvas resizing because why clutter the examples with code that's the same in every sample.
See https://webglfundamentals.org/webgl/lessons/webgl-boilerplate.html
and https://webglfundamentals.org/webgl/lessons/webgl-resizing-the-canvas.html
for webgl-utils, m3, m4, and webgl-lessons-ui.
-->
<script src="https://webglfundamentals.org/webgl/resources/webgl-utils.js"></script>
<script src="https://webglfundamentals.org/webgl/resources/m4.js"></script>
```

# WebGL Textured Cube (JS, >250 lines)



# Optimization

Vertex Array Object ★ (bad name for a modern and very cool feature)

- optimizes rendering speed and **encapsulates WebGL state**  
➡ **always use it!**
- makes WebGL look like a **modern API**
- see below

- WebGL 1 (extension always available today)

```
const ext = gl.getExtension("OES_vertex_array_object");
if (!ext) { // should never happen, extension is omnipresent!
    // tell user they don't have the required extension or work around it
} else {
    let myVAO = ext.createVertexArrayOES();
}
```

- WebGL 2

```
const myVAO = gl.createVertexArray();
```

```
// at init time
for each model / geometry / ...
    const vao = gl.createVertexArray();
    gl.bindVertexArray(vao);
    for each attribute
        gl.enableVertexAttribArray(...);
        gl.bindBuffer(gl.ARRAY_BUFFER, bufferForAttribute);
        gl.vertexAttribPointer(...);
    if indexed geometry
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
    gl.bindVertexArray(null);
```

```
// draw
gl.bindVertexArray(vao); // only 1 DrawCall !
```

```
// clean
gl.bindVertexArray(null); //Always unbind the VAO when you're done
```

# More optimizations ★

Valid for any API!

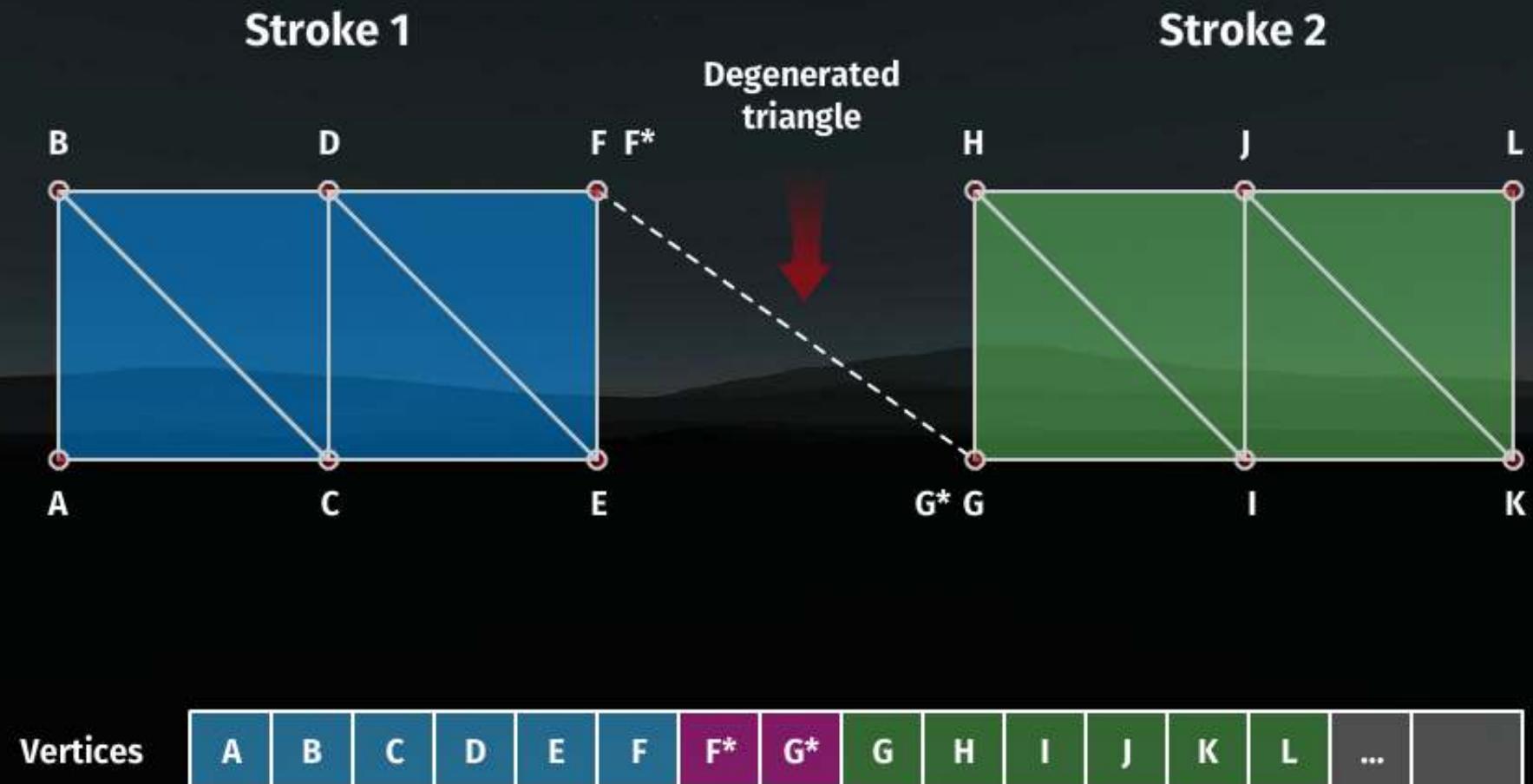
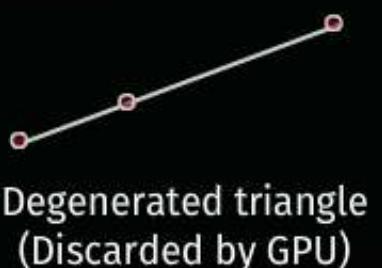
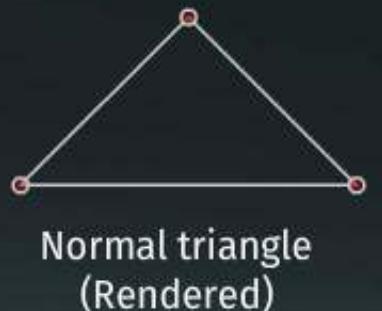
- **Profiling**
- **"Batching":**
  - Texture Atlas
  - Degenerate Triangle Strips



**npm run atlas**

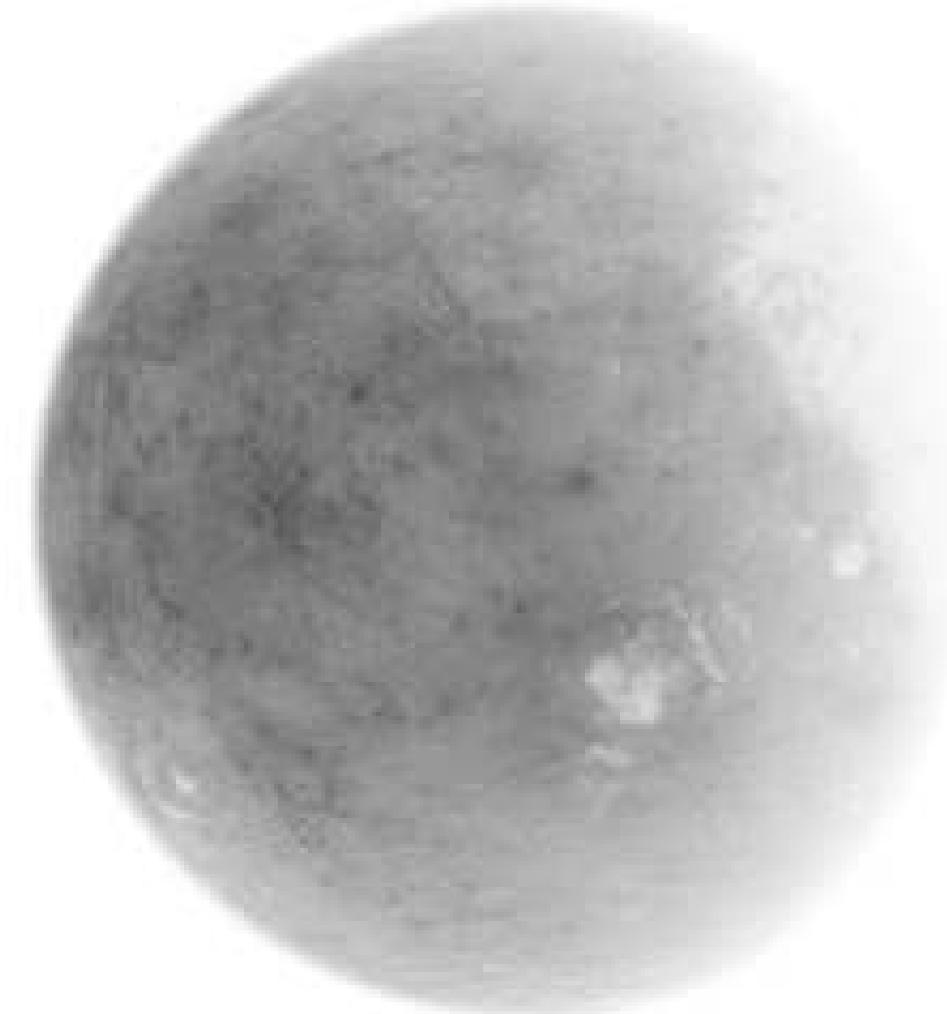


# DEGENERATED TRIANGLE



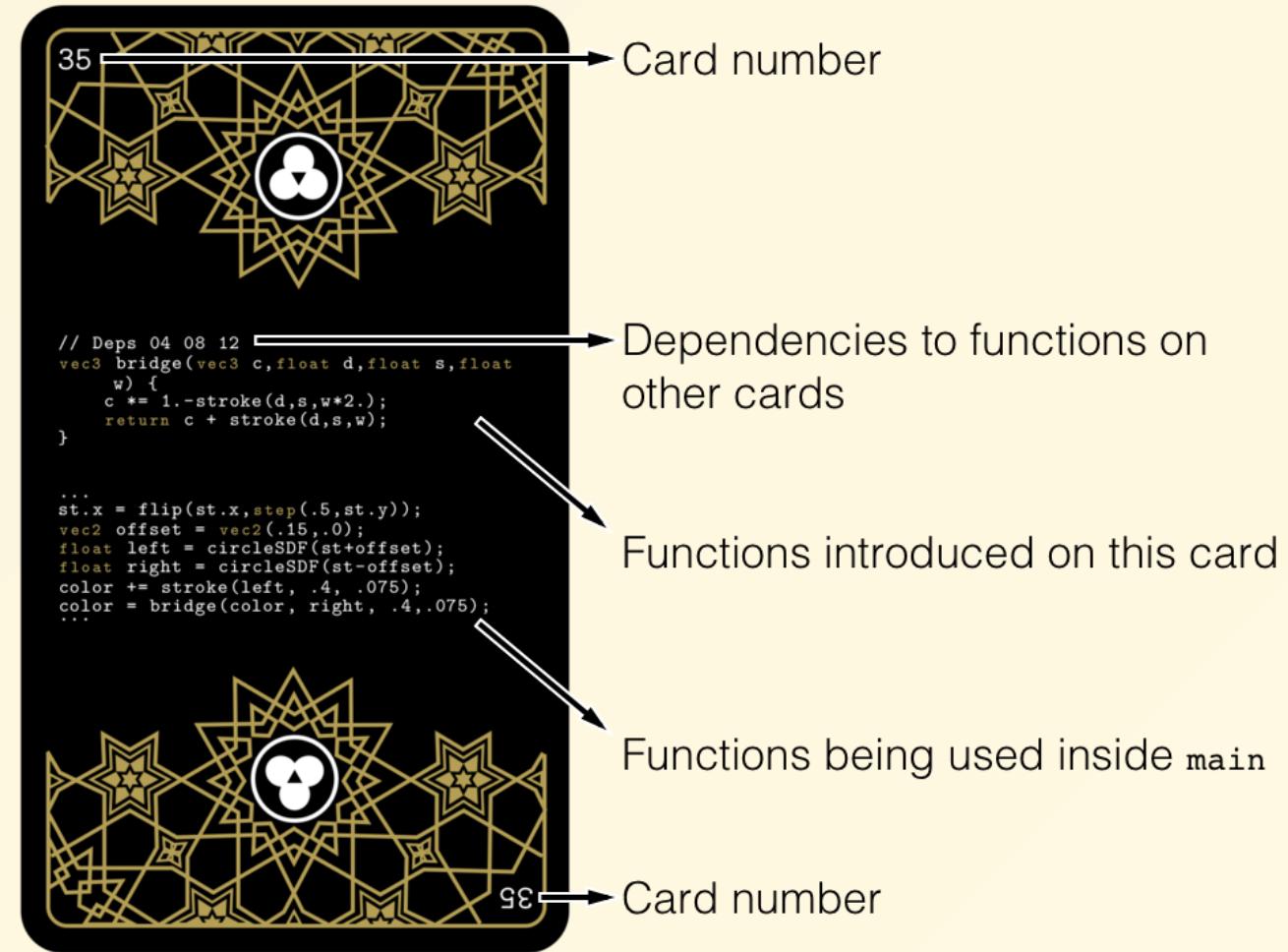
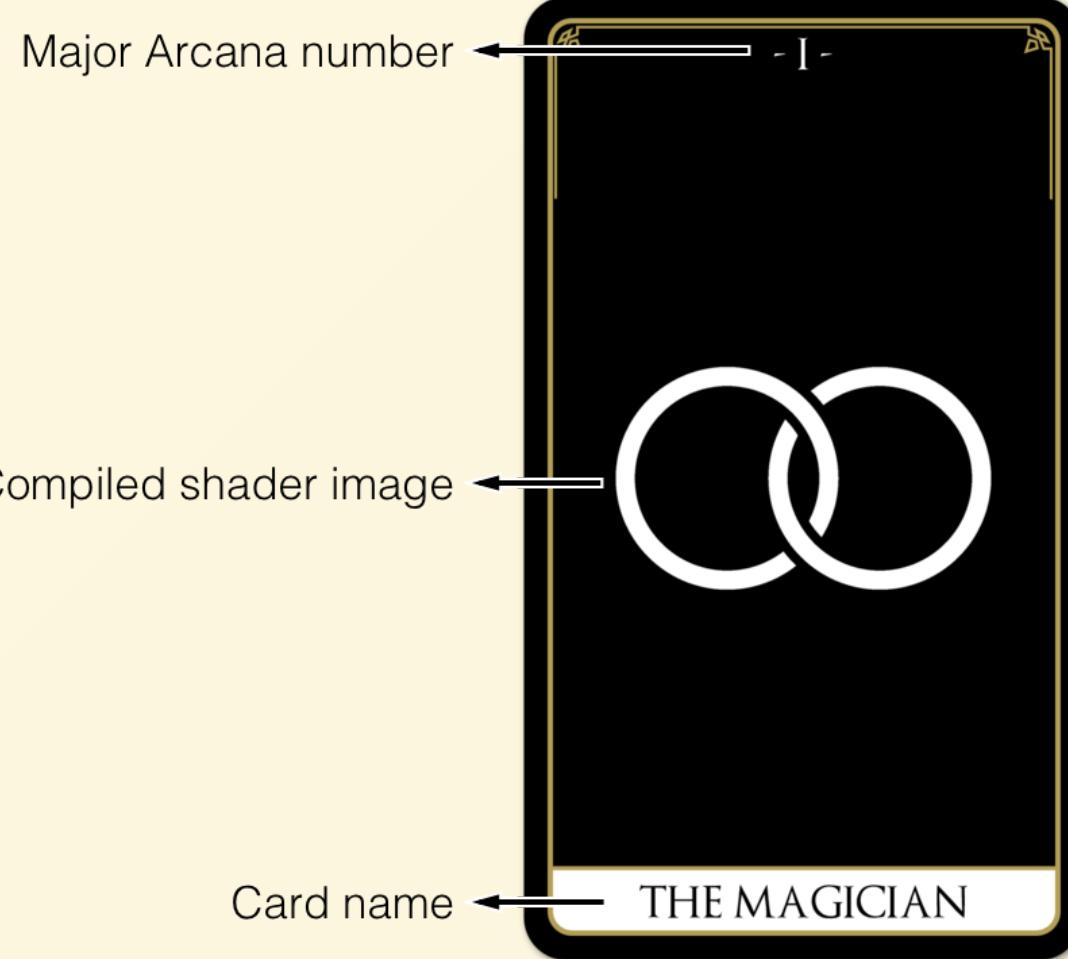
# More shaders!

# The Book of Shaders



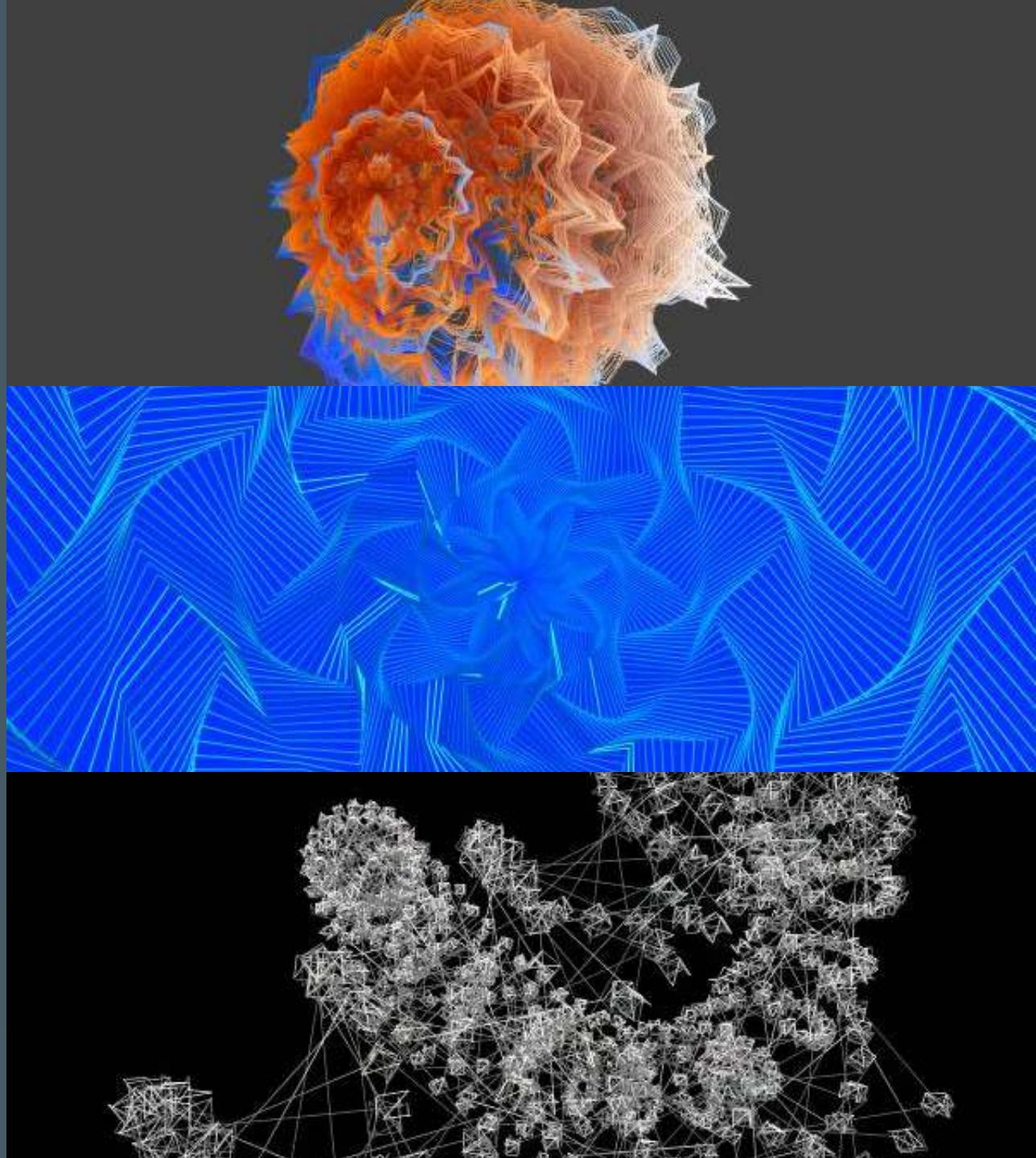
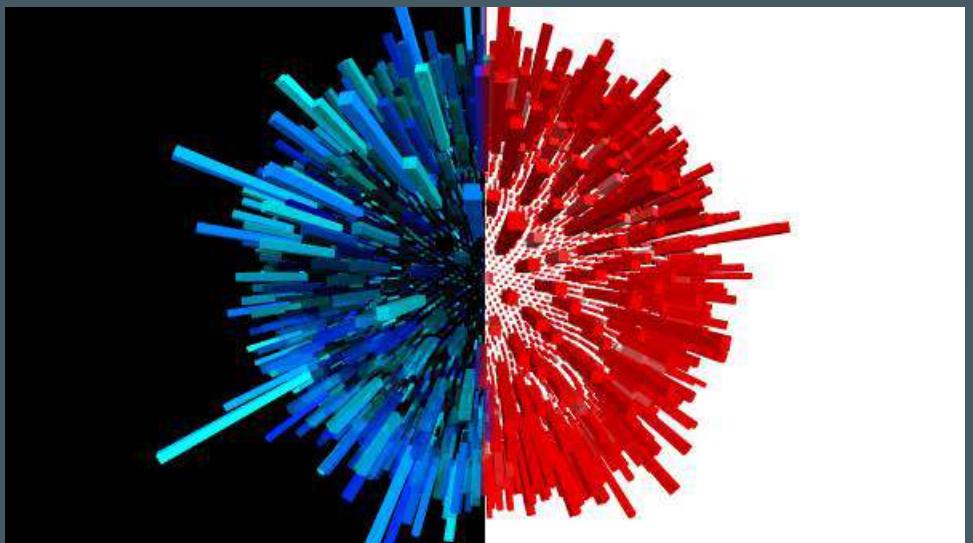
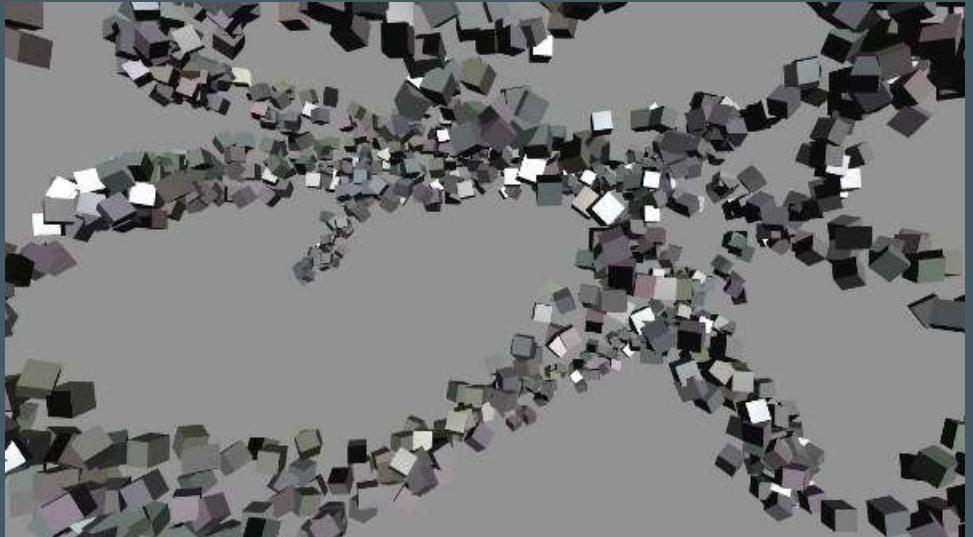
*The Book of Shaders*  
*by Patricio Gonzalez Vivo and Jen Lowe*

# PixelSpiritDeck



ShaderToy

# Vertex Shader Art



# Online shader editors

- <http://shdr.bkcore.com/>
- <https://shaderfrog.com/>
- <http://glslb.in/>
- <http://glslsandbox.com/>

# More advanced APIs

-  [WebGL 2.0](#)
  - new shaders: `in`, `out`, flexibility, `draw_buffers`, UBO and more
-  [WebGPU \(from WebGL to WebGPU\)](#)
  - generalizes **VAO** concepts to all attributes and states
-  [THREE.js](#) ★ ★ ★ ★ ★
  - high-level API: abstracts WebGL! 
  - **SceneGraph** (like OpenInventor, Unity or Blender)  
granularity: ~~buffers triangles~~ 3D objects!
  - seamless transition to WebGL2, WebGPU etc.

# WebAssembly (Wasm)



- Compile your native code for the web

{ C++, OpenGL } → [emscripten](#) → { HTML, JS (Wasm), WebGL }

- [Minimalistic example](#) summarized below

- C++

```
#include <functional>

#include <emscripten.h>
#include <SDL.h>

#define GL_GLEXT_PROTOTYPES 1
#include <SDL_opengles2.h>

//...

// an example of something we will control from the javascript side
bool background_is_black = true;

// the function called by the javascript code
extern "C" void EMSCRIPTEN_KEEPALIVE toggle_background_color() { background_is_black = !background_is_black; }

std::function<void()> loop;
void main_loop() { loop(); }

int main()
{
    SDL_Window *window;
    SDL_CreateWindowAndRenderer(640, 480, 0, &window, nullptr);
```

```

//...

loop = [&]
{
    // move a vertex
    const uint32_t milliseconds_since_start = SDL_GetTicks();
    const uint32_t milliseconds_per_loop = 3000;
    vertices[0] = (milliseconds_since_start % milliseconds_per_loop) / float(milliseconds_per_loop) - 0.5f;
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    // Clear the screen
    if( background_is_black )
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    else
        glClearColor(0.9f, 0.9f, 0.9f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw a triangle from the 3 vertices
    glDrawArrays(GL_TRIANGLES, 0, 3);

    SDL_GL_SwapWindow(window);
};

emscripten_set_main_loop(main_loop, 0, true);

return EXIT_SUCCESS;
}

```

## • HTML / JavaScript

```
<body>

    <!-- Create the canvas that the C++ code will draw into -->
    <canvas id="canvas" oncontextmenu="event.preventDefault()"></canvas>

    <!-- Allow the C++ to access the canvas element -->
    <script type='text/javascript'>
        var canv = document.getElementById('canvas');
        var Module = {
            canvas: canv
        };
    </script>

    <!-- Call the javascript glue code (index.js) as generated by Emscripten -->
    <script src="index.js"></script>

    <!-- Allow the javascript to call C++ functions -->
    <script type='text/javascript'>
        canv.addEventListener('click', _toggle_background_color, false);
        canv.addEventListener('touchend', _toggle_background_color, false);
    </script>

    <!-- Describe what the user is seeing -->
    <p>Click the canvas to change the background color.</p>
    <hr>
    <p>Minimal example of animating the HTML5 canvas from C++ using OpenGL through WebAssembly.</p>
    <p>Source code: <a href="https://github.com/timhutton/opengl-canvas-wasm">https://github.com/timhutton/opengl-canvas-wasm</a></p>

</body>
```

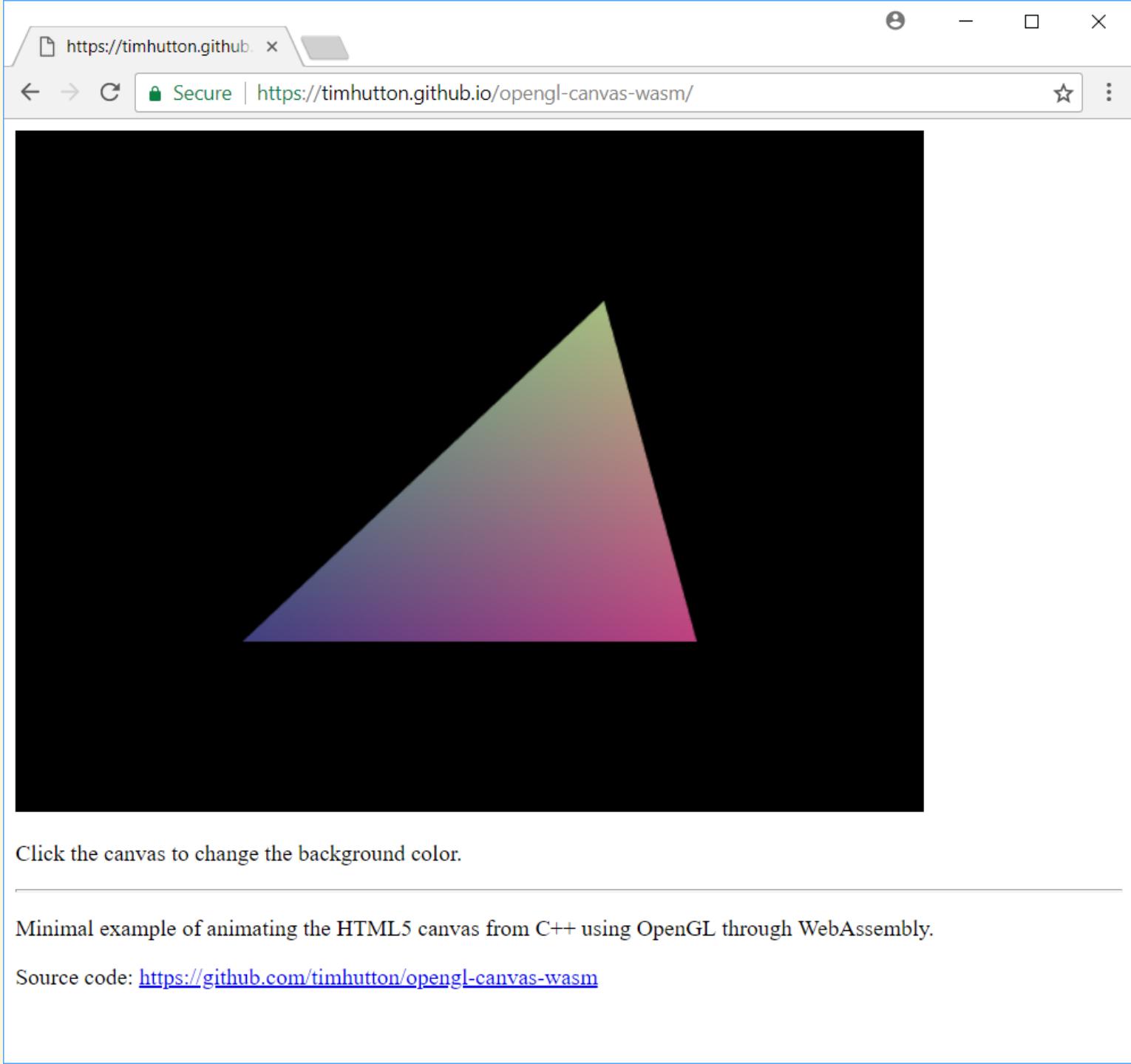
# Compilation

- Install [Emscripten](#)
- Generate `index.js` and `index.wasm`:

```
emcc main.cpp -std=c++11 -s WASM=1 -s USE	SDL=2 -O3 -o index.js
```

- Open `index.html` (using server)

# Result



Click the canvas to change the background color.

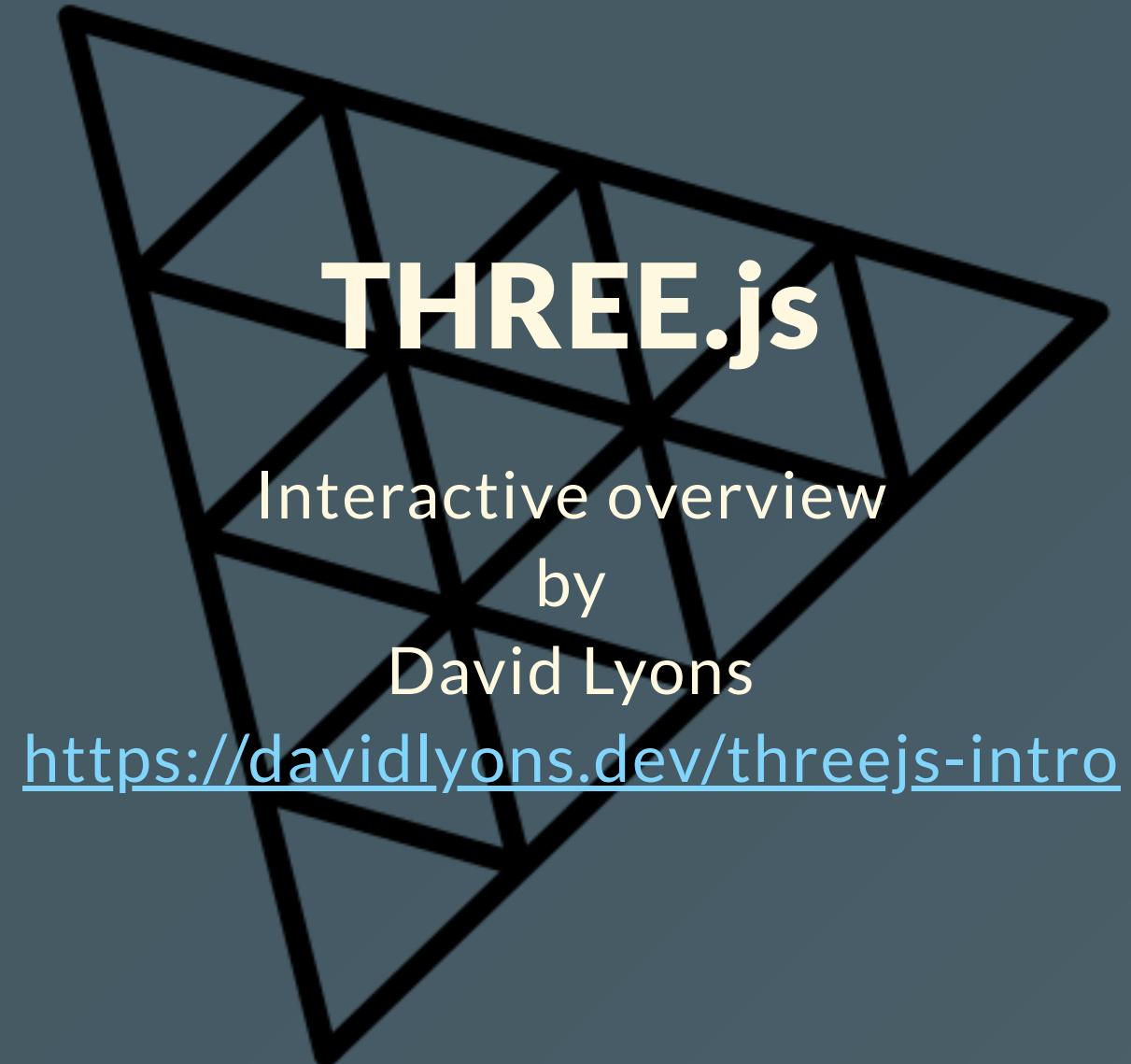
Minimal example of animating the HTML5 canvas from C++ using OpenGL through WebAssembly.

Source code: <https://github.com/timhutton/opengl-canvas-wasm>

“ I implemented WebGL in Chrome and my name is on the spec so I have some clue how it works ”

“ TL;DR  
**If you want to get stuff done use three.js.** ”

Gregg Tavares (@greggman)



# THREE.js

Interactive overview

by

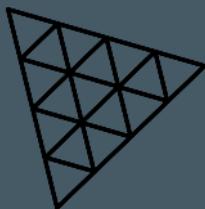
David Lyons

<https://davidlyons.dev/threejs-intro>

# Intro to WebGL with Three.js

Go to slide:  Go

# THREE.js Manual



★ BEST Guide ★

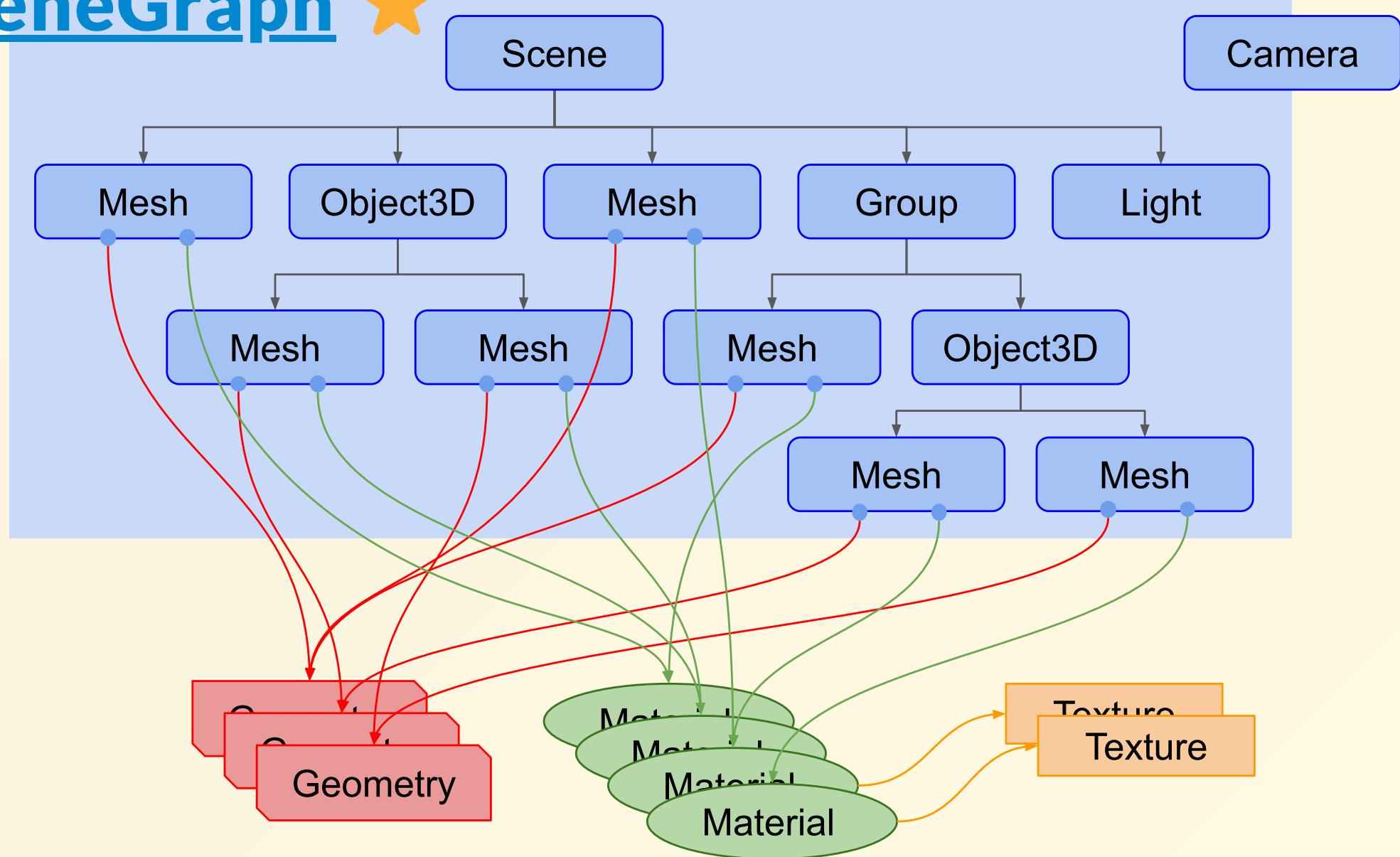
<https://threejs.org/manual/#en/fundamentals>

Complete and simple guide



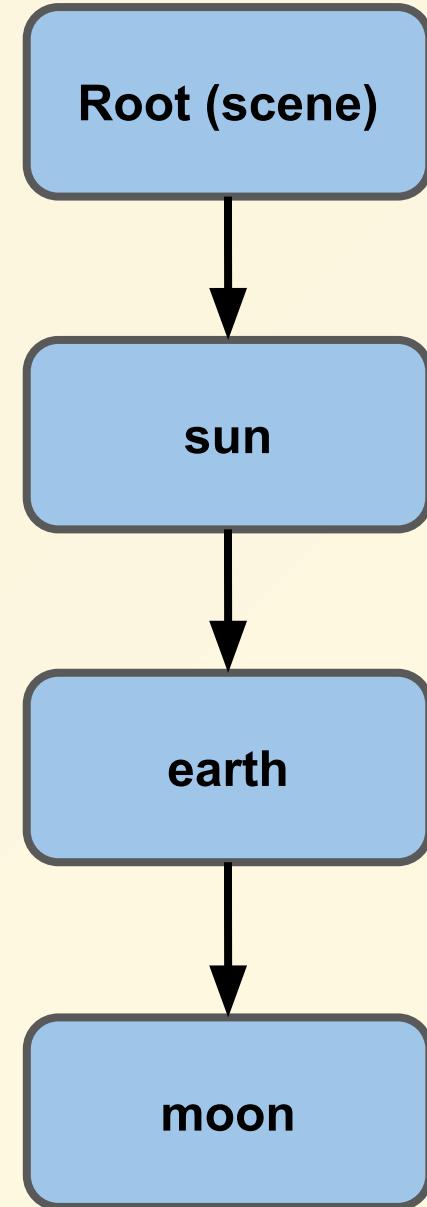
Renderer

# SceneGraph



- we handle **3D objects** instead of buffers
  - higher level, easier, more intuitive
- each scene is organized as a **hierarchy** of objects
  - hence the term "**scene graph**"
- allows to combine **local transforms** into global transforms
  - ex: solar system (see below), wheels of a car
- **rendering API abstraction**
  - ex: seamless transition from WebGL to WebGPU
- **scene graph optimizations**
  - batching
  - smart update of 3D objects

# Solar system example





# THREE.js Manual (excerpts)

- load and handle **glTF / glb** models
- how to create a game
  - presentation (**progress bar**)
  - code **architecture** notions
  - **keyboard** input
  - **glTF animations**



# Setup

**in order to "Build"**

How to run the examples locally

# Setup

- THREE.js is a **library**, **NOT** a standard API like WebGL
- THREE.js abstracts WebGL 1, WebGL 2 and WebGPU
- We need to **import** its modules before coding:
  - 1 either using **CDN** ([Content Delivery Network](#))
    - **zero setup**: allows **quick tests, without installation**
  - 2 or through a full **installation** (via [Node.js](#))
    - allows complete access to all resources, but introduces a complex toolkit ([npm](#), [webpack](#), [rollup](#) etc.)
  - ⚠ zip download **NOT** recommended (complex dependencies)

# Zero Setup: using a CDN

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <title>three.js</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">
  </head>

  <style>
    html, body { margin: 0; padding: 0; overflow: hidden; }
  </style>

  <body>

    <script type="importmap">
    {
      "imports": {
        "three": "https://cdn.jsdelivr.net/npm/three@0.169.0/build/three.module.js",
        "three/addons/": "https://cdn.jsdelivr.net/npm/three@0.169.0/examples/jsm/"
      }
    }
    </script>

    <script type="module">

      // Example of hard link to official repo for data, if needed
      const MODEL_PATH = 'https://raw.githubusercontent.com/mrdoob/three.js/r169/examples/models/gltf/LePerrySmith/LePerrySmith.gltf';

      import * as THREE from 'three'

      import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
      import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js';

      // INSERT CODE HERE

    </script>
  </body>
</html>
```

- **Hello Cube** (modern ES6+ version, using `const`, `let` and `=>`)

```
const scene = new THREE.Scene();
const aspect = window.innerWidth / window.innerHeight;
const camera = new THREE.PerspectiveCamera( 75, aspect, 0.1, 1000 );
const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshNormalMaterial();
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );
camera.position.z = 5;

const render = () => {
    requestAnimationFrame( render );
    cube.rotation.x += 0.1;
    cube.rotation.y += 0.1;
    renderer.render( scene, camera );
};

render();
```

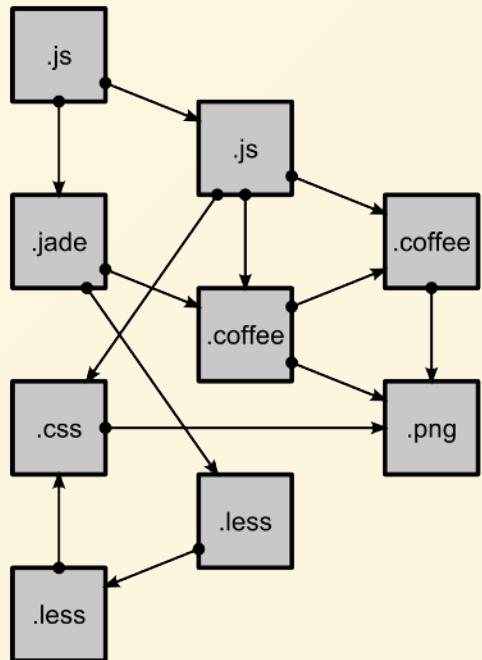
## 2 Full setup using NPM

- install Node.js + install npm

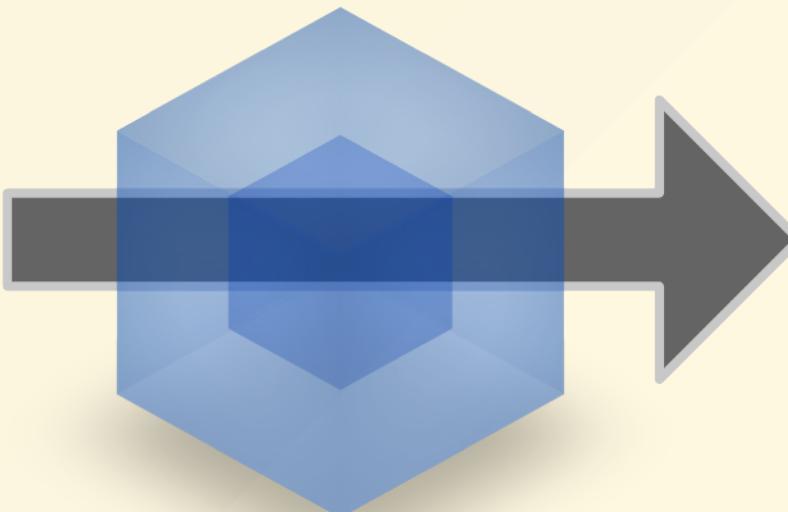
```
sudo apt install nodejs  
curl -L https://npmjs.org/install.sh | sudo sh
```

See below

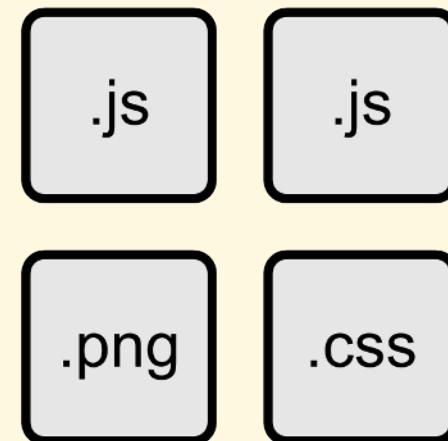
# Bundlers



modules  
with dependencies



**webpack**  
MODULE BUNDLER



static  
assets

# Manual installation 😠

- ⚠️ **avoid manual installation!**
  - shown here for educational purposes
  - ➡️ **use three vite template instead: automatic installation**
- install vite

```
npm install --save-dev vite
```

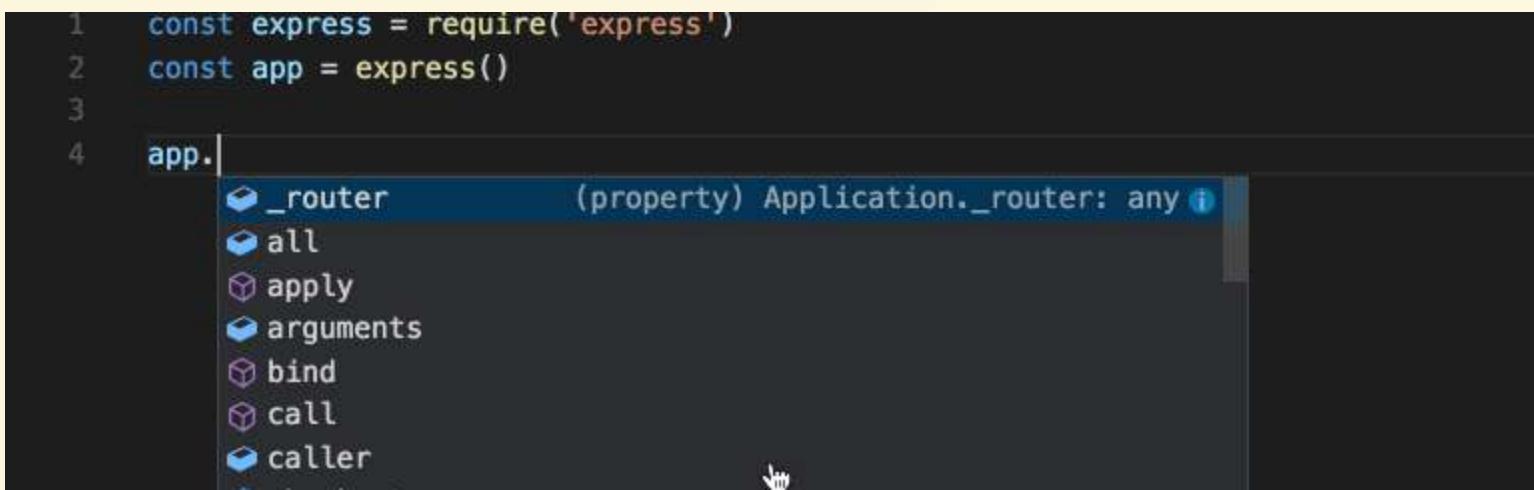
- install **three**

```
npm install three
```

# THREE.js IntelliSense

- easy access to the **documentation** of THREE.js classes
- **autocomplete**
- **type checking**, even in JavaScript

```
npm install @types/three --save-dev
```



A screenshot of a code editor showing a code snippet and an autocompletion dropdown. The code is:

```
1 const express = require('express')
2 const app = express()
3
4 app.|
```

The cursor is at the end of 'app.' in line 4. An autocompletion dropdown is open, showing suggestions:

- \_router (property) Application.\_router: any
- all
- apply
- arguments
- bind
- call
- caller

# Automatic installation

## THREE.js with "batteries included"



# THREE Vite boilerplate + =

Preconfigured environment (allows to test all official examples)

[https://github.com/fdoganis/three\\_vite](https://github.com/fdoganis/three_vite) ★

```
git clone https://github.com/fdoganis/three_vite.git
```

```
cd three_vite
```

```
npm install
```

Run with `npm run dev` or use `F5` in VS Code

Open `http://localhost:5173` in your browser

# Boilerplate alternatives

-  [\*\*Vite\*\*](#) (uses Rollup and [esbuild](#) for speed) 
  - <https://github.com/j13ag0/vite-GLTFloader-test>
-  [\*\*Rollup\*\*](#) (the bundler by THREE developer team)
  - <https://github.com/Mugen87/three-jsm>
-  [\*\*Parcel\*\*](#) ([tutorial](#)) (boilerplate)
  - [https://github.com/fdoganis/three\\_parcel](https://github.com/fdoganis/three_parcel)
  - <https://github.com/franky-adl/threejs-starter-template> ([article](#))
-  [\*\*webpack\*\*](#)
  - <https://github.com/edwinwebb/three-seed/>

# Bundler wars

-  [Rollup](#) vs  [webpack](#) vs  [Parcel](#) [\(Comparison\)](#).
-  [Vite](#) vs  [Parcel](#) vs  [esbuild](#)  [\(Comparison\)](#).
-  If you don't have a favorite bundler, use:
  - **Vite** (simple, very fast and recent, might be used by THREE) 
  - **Rollup** (quite simple and fast, used by THREE)
  - **Parcel** (too simple for THREE modules)
  - **webpack** (flexible but complex and slow)
  - **esbuild** (the fastest!)

# Basic THREE.js concepts

# Let's build a solar system

- full tutorial [here](#) ←
- create a webpage with a canvas
- create a scene with lights and meshes
  - understand object hierarchy
    - add vs attach
- render the scene using a camera and renderer
- animate the scene using requestAnimationFrame
  - or setAnimationLoop
- see below ↴

three.js / editor

保護された通信 | <https://cdn.rawgit.com/mrdoob/three.js/dev/editor/index.html>

File Edit View Examples Help

New

# THREE.js Editor

Import • <https://threejs.org/editor/> ★

Export Geometry  
Export Object  
Export Scene  
Export GLTF  
Export OBJ  
Export STL  
Publish

objects 2 vertices 108 triangles 36

CC BY-SA 4.0 | 2024 ©Fivos Doganis

basic 3D model edition

basic material edition

simple 3D format conversion

use THREE.js editor to create a hierarchy

- sun ☀
- earth 🌎
- moon 🌙

SCENE PROJECT SETTINGS

- Camera
- Scene
- Box 2
- DirectionalLight 1

Background

Fog NONE

OBJECT GEOMETRY MATERIAL

Type DirectionalLight

UUID EC718DB3-937A-4 NEW

Name DirectionalLight 1

autosave r87dev

112

# Advanced THREE.js

# Advanced THREE.js features

- THREE.js API still allows low-level (WebGL) access, using :  
**BufferGeometry** + **RawShaderMaterial**
  - example: convert a terrain texture into 3D
    - see [webglacademy](#)
- **Custom Geometry, indexed, dynamic**

→ you can still create your own WebGL shaders and custom geometry in THREE.js! ★ 🎉

But most of the time the existing classes should be enough!

# Good coding practices

- Modernize JavaScript code!
- WebGL best practices
- WebGL2 best practices
- Reduce Draw Calls
- use OffscreenCanvas and JavaScript workers
  - Web Workers in the Real World
  - Moving a Three.js-based WebXR app off-main-thread
    - OffscreenCanvas still not available on iOS 😢

# Helpers

- Tutorial

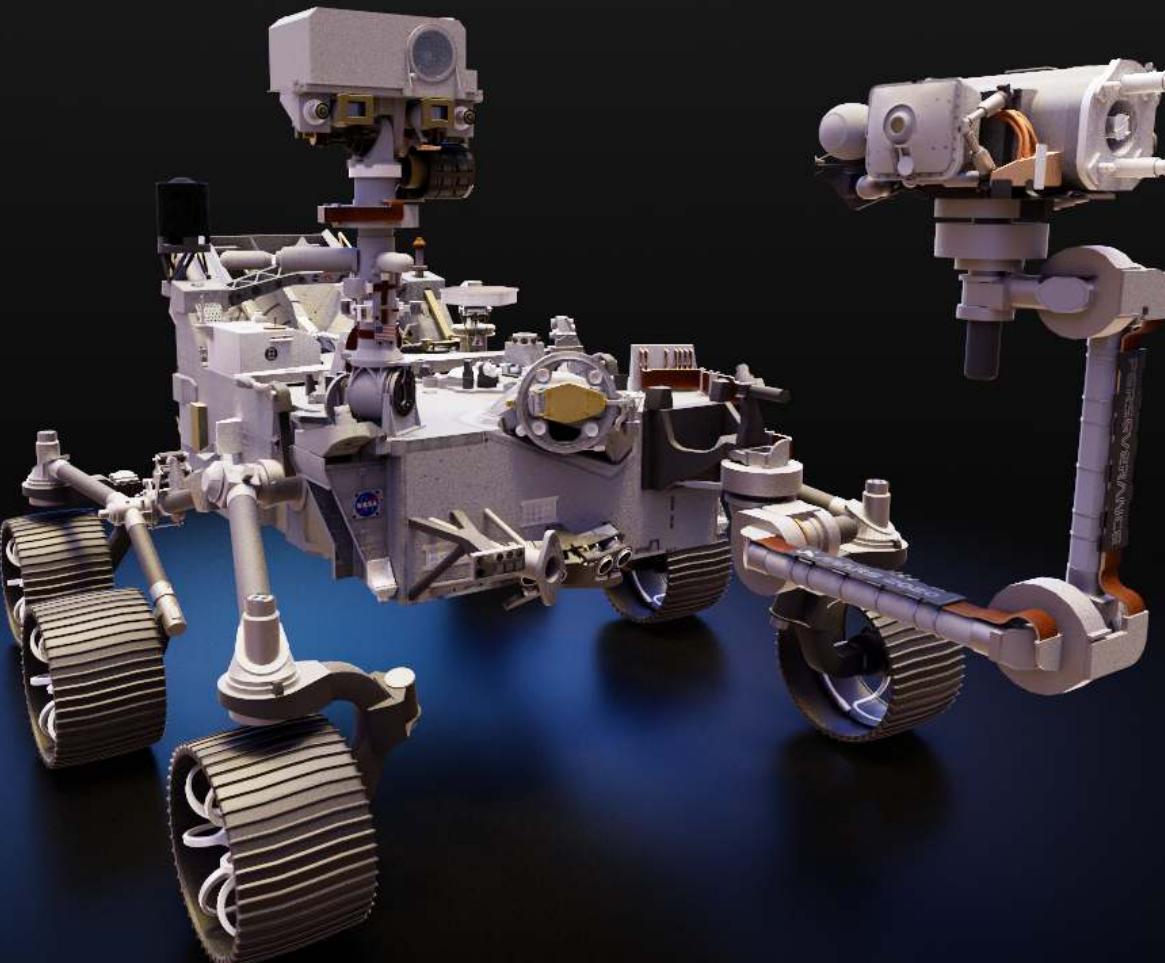
# Debugging tips



- <https://threejs.org/manual/en/debugging-javascript.html> ★
- <https://discoverthreejs.com/tips-and-tricks/>
- Rendering on demand (onTouch) + **lil-gui** (dat.gui)  
<https://threejs.org/manual/en/rendering-on-demand.html>
- lil-gui: Nice [UI example](#) to modify fog parameters interactively
- **Debugger:** <https://github.com/spite/ThreeJSEditorExtension>
- <https://github.com/threejs/three-devtools> for [Firefox](#), [Three-tools by BACE](#) ([Chrome et Firefox](#)), and [Spector.js](#)

# Extra features

- [Troika \(Doc\)](#)
- [Advanced camera](#)
- [3D Tiles \(NY Times\)](#)
- [Optimized post-processing](#)
- [BVH \(Bounding Volume Hierarchy\)](#)
- [GPU PathTracer \(BVH\)](#)
- [Official THREE.js links](#)



# Physics (official examples)

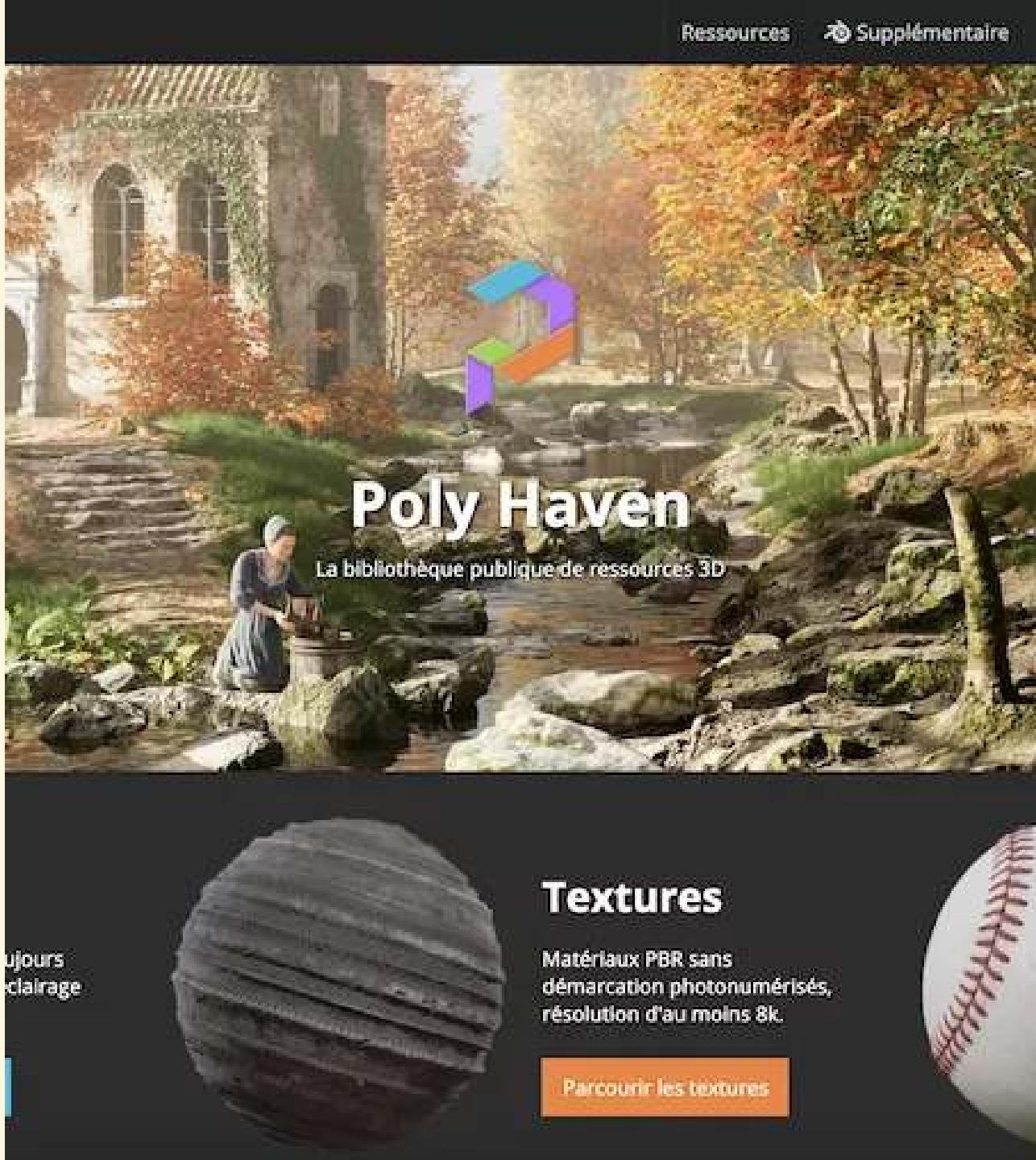
- ~~Cannon.js~~  / [Cannon-es](#)  : JS port of [Bullet Engine](#) ([Tutorial](#)).
  - [Demo!](#)   
- [Ammo.js](#) : conversion of Bullet (C++) to JS (Wasm) ([Tutorial](#)).
  - very fast but can be difficult to debug
- [Rapier](#) : Rust physics engine converted to JS (Wasm) ([Tutorial](#)).
- [Oimo](#) : Haxe physics engine converted to JS (Wasm) ([Tutorial](#)).
- [Box2D](#) : sometimes 2D is enough! (planar movements) ([Tutorial](#)).
- [Phy](#) : a wrapper to rule them all, including [PhysX](#), [Havok](#) and [Jolt!](#)

# 3D Models

- prefer glTF / glb format → Official THREE.js Tutorial ★
  - optimize your models using glTF-Transform
- SketchFab
- Low poly marketplace
- Poly pizza, backup of Google Poly 💀 : here
- TurboSquid
- Sketchup 3D Warehouse
- Polyhaven
- Legal? (ripped)

# Textures

- PolyHaven
  - [HDRIs](#)
  - [Textures](#)
- [Unsplash](#)
- [Pexels](#)



# THREE.js exercises

# THREE.js exercises

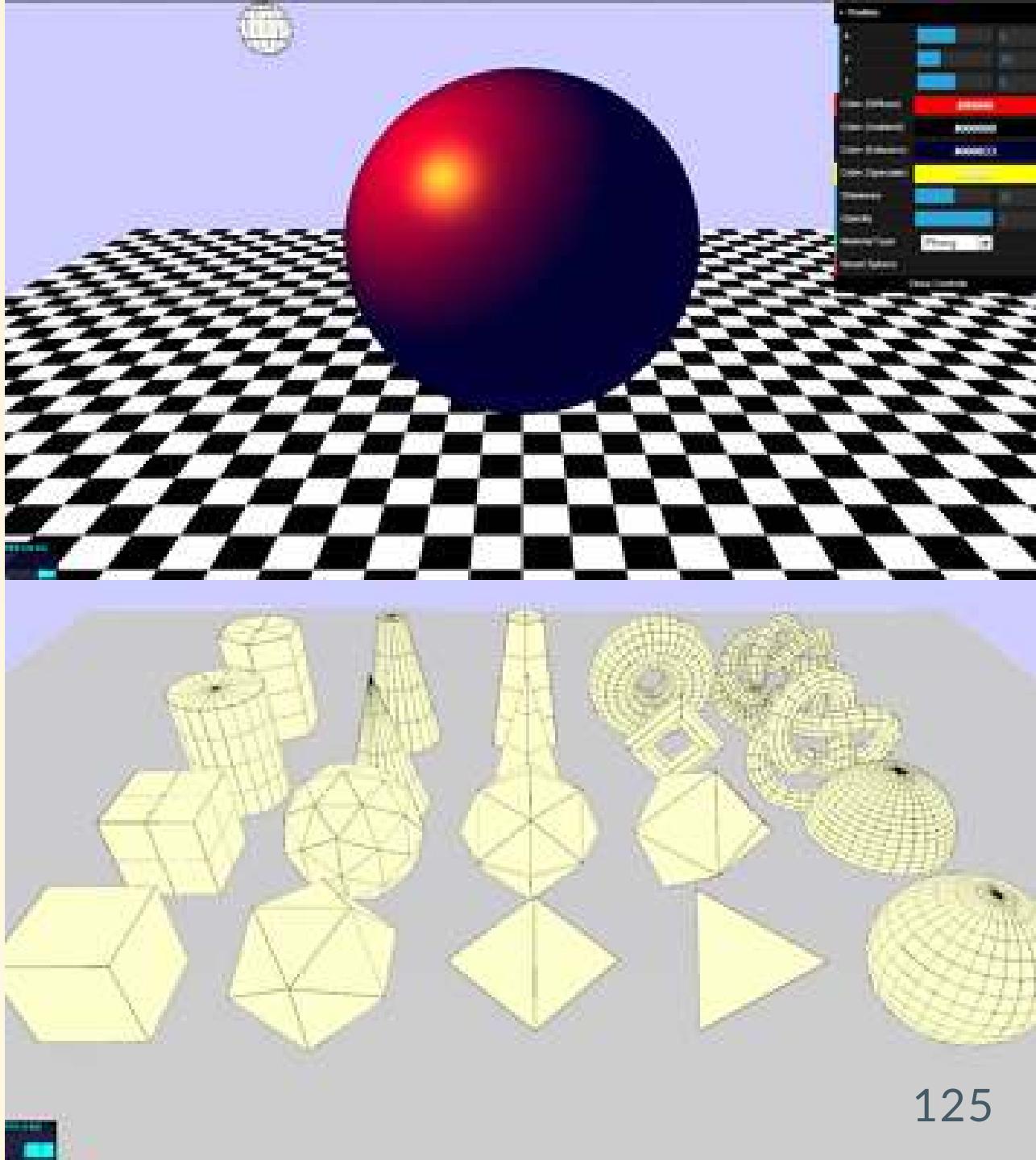
- Reminder: [THREE.js Manual](#) ★
- [Karim Maaloul](#)
  - practice:
    - [animals](#)
    - [aviator](#) ★
    - [aviator 2 \(update\)](#) ●●
    - [red bull](#)



# THREE.js Projects

# Examples

- Official examples
  - always up to date!
  - recommended!
    - ! THREE.js API changes **very** often!
    - JavaScript API too...
- Lee Stemkoski
  - old examples
    - code needs updating



# Project ideas

# Fun

- Adapt existing code or games to use THREE
  - board games
  - "Flash" games
  - classic (video) games (mini-games, party games, sports)
  - add sound, and physics
- [Christmas Cannon](#) ★

# More project ideas

- implement a complex effect
  - raytracing, SSAO, OIT, NPR
- image processing, OpenCV
- physics
  - pool, bowling, domino, jenga, stacks
- particles: text / image to cloud of cubes
- 3D WebRTC pong (other WebRTC examples here, here and here)
- animal crossing
- personal website, interactive CV: HTML layout, THREE.js Journey 128

# Healthcare Projects

- Volume rendering
- Cinematic rendering
- Gaussian splatting
- Transparency
  - OIT
  - Depth Peeling
- Organ hide / show / clip / manipulate
- DICOM / CT Scan

# Healthcare Projects 2

- Animations
  - UI
  - IK skeleton
- Annotations
- UX
  - Haptic feedback
  - Text
  - Positional Audio

# Healthcare Projects 3

- Communicate / Educate / Train / Guide
  - Load models
  - Z anatomy / Zygote body viewer
    - Performance
      - Show difference in fps
      - BVH / Octree / Tiles / LOD / Load / unload
  - Medical Quizz

# Healthcare Projects 4

- Staying Alive
  - In AR?
  - Use face or image
- Step by step surgical operation
- Magic mirror see through
  - Skeleton
  - Back face rendering
  - Plastic surgery

# Healthcare Projects 5

- Simulation
  - Stenosis
  - Beating heart
  - Mesh Deformation
  - Soft tissues
  - Cut organ
    - "Organ Ninja"

# Healthcare Projects 6

- Operation Game (Dr Maboul)
  - Surgeon Simulator
  - Blood
  - Splat
- Air flow virus spread
- Virtual Twin / 3D reconstruction / scan / Scaniverse : patient / surgeon / room
  - Mesh / Gaussian Splats
  - Avatars

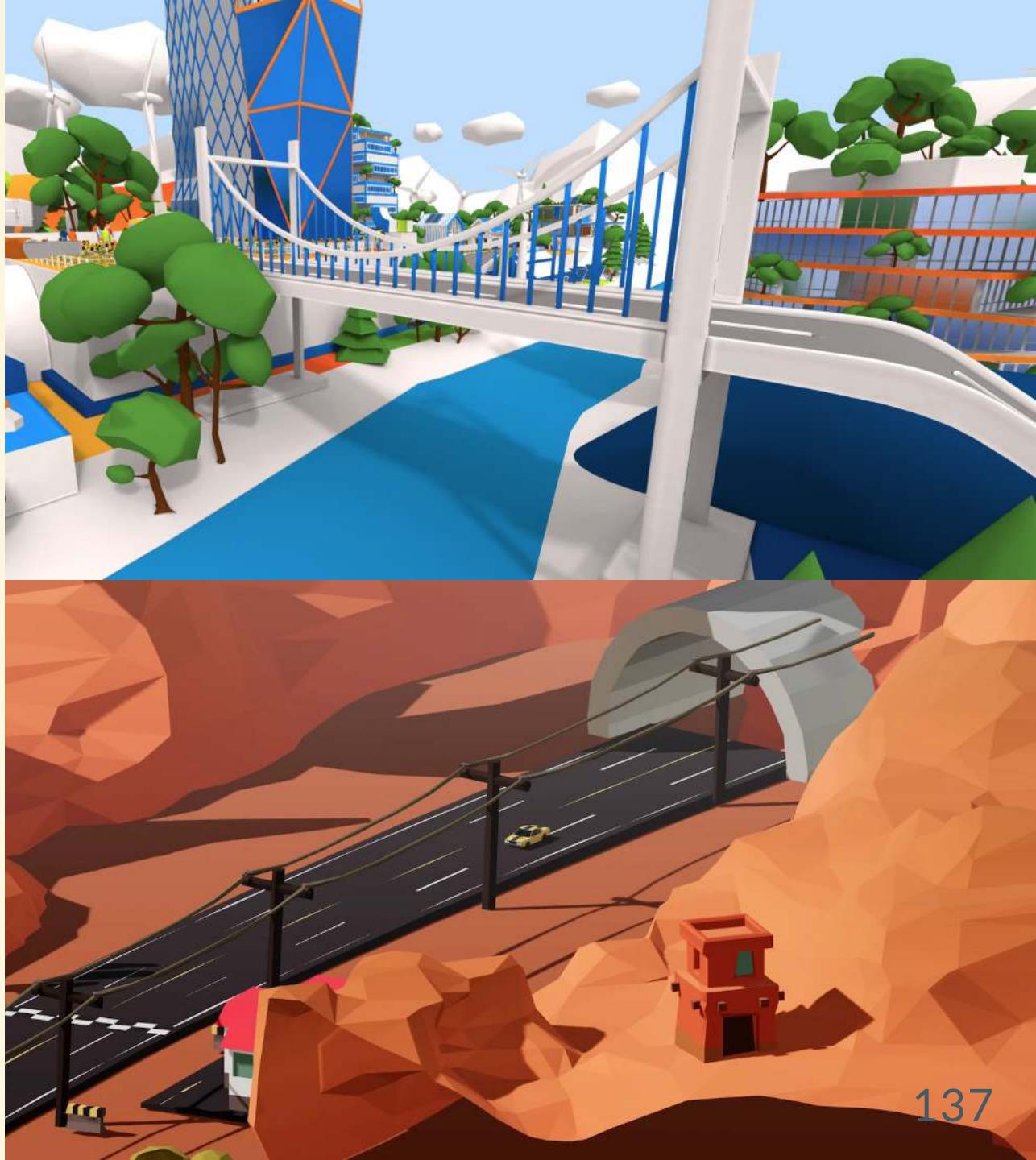
# Healthcare Projects 7

- Instrument manipulation
  - UI
  - [GraspUI](#)
  - Pillbox AR
- Collab
- 360 video
- Hospital twin
  - Indoor GPS

# Wrappers

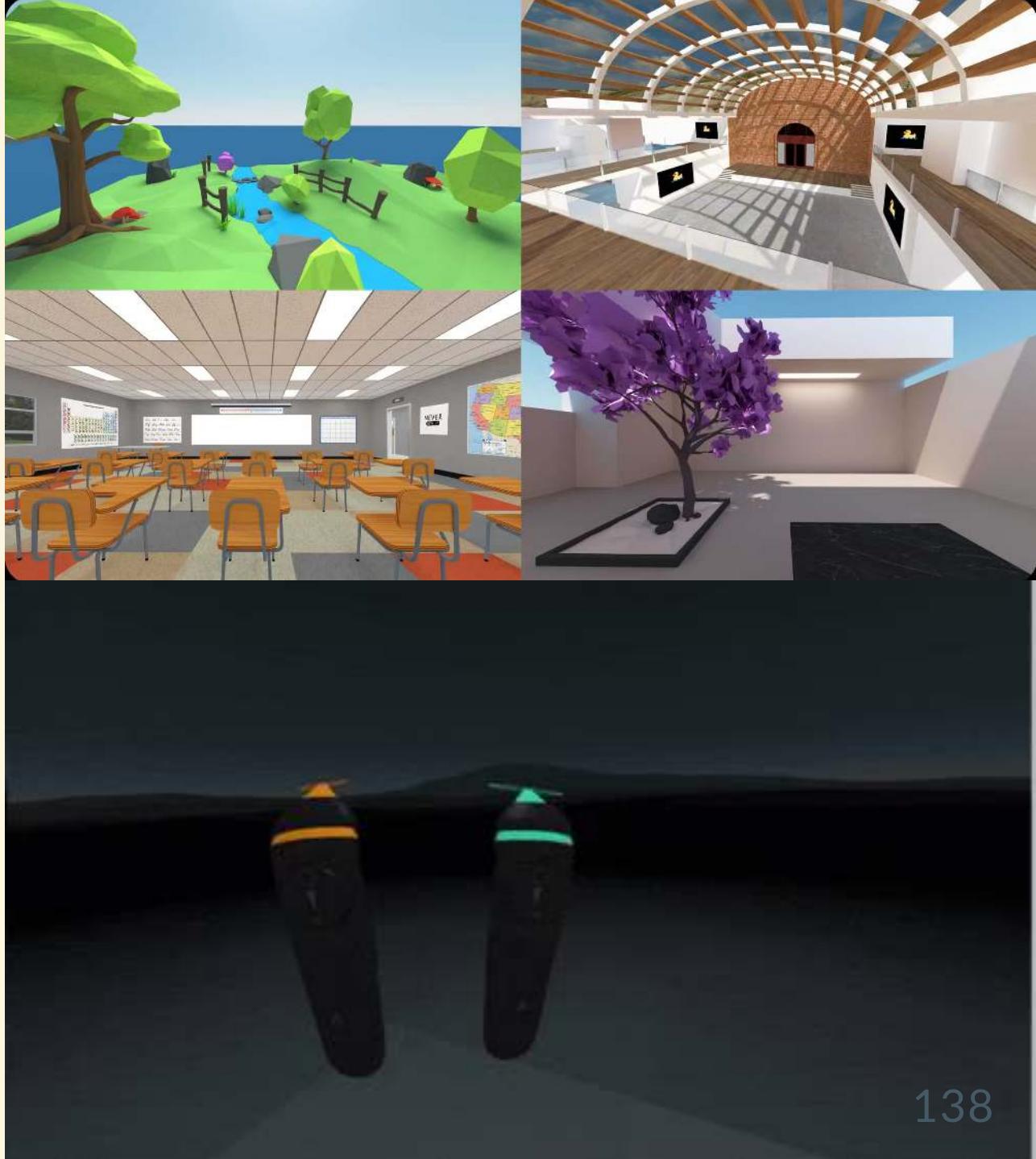
# React Three Fiber (R3F)

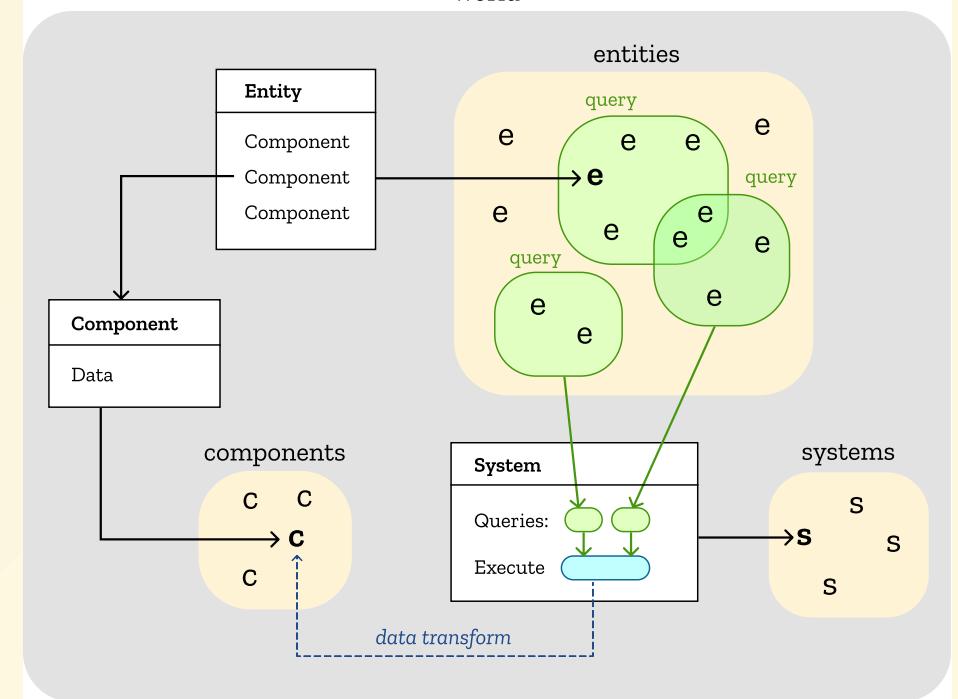
- Great for "pro" web sites
- [Tutorial](#)
- Demos
  - [EDF electric days](#)
  - [Racing game](#)
- [React Native](#)



# A-Frame

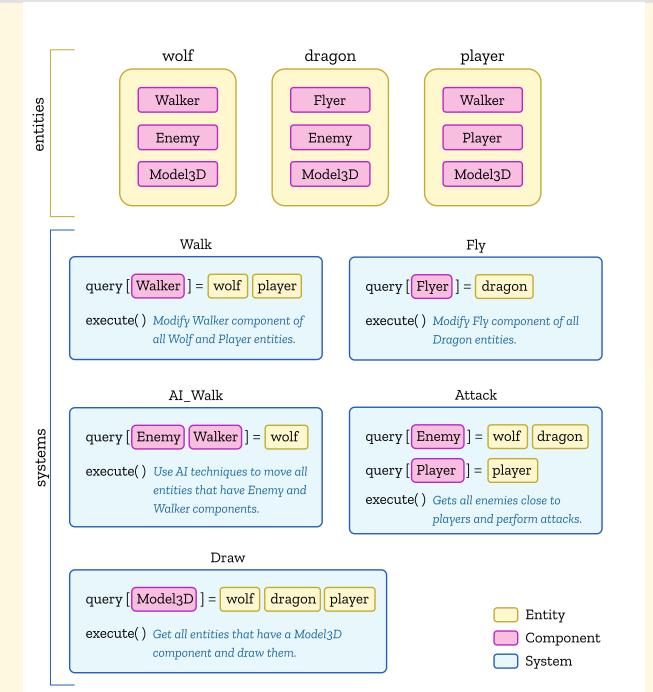
- Declarative wrapper (tags)
- Extremely simple to use (very high-level)
- Modular and extensible (**ECS**: [Entity Component System](#))
- Ideal for AR and VR
  - [Mozilla Hubs](#)
  - [A-Painter](#)





# ECS

- [ECSY](#) 💀
  - articles: [Mozilla](#), [Medium](#)
  - [ECS in 99 lines of code](#)
- [bitECS](#) ★
- [WolfECS](#)
- [Ape-ECS](#)
- [nopun-ecs](#)
- [ECS @ Apple \(Video 25'\)](#) ★
- used in [Overwatch!](#)



# Tests

```
const THREE = require('three');
const assert = require('assert');

describe('The THREE object', function() {
  it('should have a defined BasicShadowMap constant', function() {
    assert.notEqual('undefined', THREE.BasicShadowMap);
  }),

  it('should be able to construct a Vector3 with default of x=0', function() {
    const vec3 = new THREE.Vector3();
    assert.equal(0, vec3.x);
  })
})
```

# The End!