

WebGL SE Survey

Zhen Zhang, Xingan Wang

April 10, 2016

Abstract

The survey is trying to answer the questions based on collected facts and statistics:

1. What is the best practice in programming WebGL programs?
2. What is the common pitfalls in programming WebGL programs?

Contents

I	Introduction	5
	WebGL and OpenGL standard	5
	Evolving of the standards	5
	Information in the standards	5
	WebGL	5
	OpenGL ES	5
	Example Implementation	8
	Conformity status of popular implementations	8
	Desktop browsers	8
	Mobile browsers	8
	More updated information	9
	Online API documentation	9
	Sources	9
	Interface structure	10
	MDN	10
	MSDN	10
	Usability	10
	Edge cases and correctness	10
	Compatibility, security and performance	10
	MDN	10
	MSDN	10

Ecosystem	11
Books	11
Tutorials	12
Misc	13
Applications	14
Examples	14
Chrome Experiments	14
Error and warnings sampling	14
 II Troubles and Trouble-Shooting	 15
Overview	15
Categories	15
GLSL errors	15
 Compiled Information from MDN	 18
Source of bugs	18
canvas element	18
gl Object	18
Shader's source	19
Attribute names	19
Array	20
Things to avoid	20
 Compiled Information from Stackoverflow	 21
Examples	21
Undeclared identifier	21
Context management	21
Resource loading order	21
API usage error	22
Shader error	22
Threejs	23
Others	23
Type errors	23
Matrix related	23
 Compiled Information from three.js	 23
Overview	23
Details	24
threejs-pull-1602	24
threejs-issue-5269	26
threejs-issue-5421	27
threejs-issue-6956	28
threejs-issues-83	28

threejs-pull-1602	29
Compiled Information from applications	29
fireworks-webgl	29
medusae	29
Trouble-shooting WebGL code	30
<i>Professional WebGL Programming: Developing 3D Graphics for the Web</i>	30
<i>An Introduction to WebGL Programming</i>	30
<i>Beginning WebGL for HTML5</i>	30
Misc	31
Statistics	31
 III Library Support	 33
Overview	33
Popular libraries	33
Abstractions	33
Why libraries?	33
Case study: gl-matrix	34
APIs	34
Common utilities	34
3x3 Matrix	34
Library structure	34
Possible problems	34
Case Study: Threejs	34
API samples	34
Library structure	39
 IV Tool Support	 41
Exclusive Debug tools	41
WebGL Insights	41
WebGL-Inspector	41
webgl-debug.js	44
WebGL Linter	44
Browser Support and IDE Support	44
Browser	44
Chrome Canvas Inspector	44
Firefox Shader Editor	44
IDE	44

JetBrain - WebStorm	45
WebGL Studio	45
Three.js editor	45
3Dmol.js	45
Blend4Web	45
V Conclusion	46
Best practices in WebGL development	46
General ones	46
Technical ones	46
Future of WebGL tooling	46
Static analysis technique	46
Verified libraries	46
Integration	47

Part I

Introduction

WebGL and OpenGL standard

Evolving of the standards

The publication and draft dates of WebGL specification:

- Version 1.0, 10 February 2011
- Version 1.0.1, 27 January 2012
- Version 1.0.2, 01 March 2013
- Version 1.0.3, 27 October 2014
- Version 2.0, 19 February 2016 (latest draft)

NOTE: Version 1.x are based on OpenGL ES 2.0; Version 2.x are based on OpenGL ES 3.0.

NOTE: The 2.0 draft spec [provided here](#) should be read as an extension to the WebGL 1.0 specification. It will only describe the differences from 1.0.

Information in the standards

WebGL

In [WebGL spec](#), it introduces *Context Creation* and *Drawing Buffer Presentation*, *WebGL Resources* and *Security* only briefly. The major parts are: *DOM Interfaces* and *Differences with OpenGL ES 2.0*.

In DOM interfaces, the types and various object interfaces are introduced, in which the `WebGLRenderingContext` is the biggest one. The IDLs are presented here, and its intended semantics are described. However, it refers OpenGL ES 2.0 frequently and don't give a lot of information which appears in the OpenGL spec.

Here is a sample spec:

OpenGL ES

The [OpenGL ES 2.0 spec](#) is a rather detailed specification. Some implementation contrives are explained. So it is necessary for understanding the browser implementation.

Here is a sample spec:

5.14.5 Buffer objects

Buffer objects (sometimes referred to as VBOs) hold vertex attributes.

void glBindBuffer(GLenum target, WebGLBuffer? buffer)

Binds the given WebGLBuffer object to the given binding target. If any buffer currently bound to this target is unbound. A buffer can only be bound to the ELEMENT_ARRAY_BUFFER target in its lifetime. An unbound buffer's current binding will remain untouched.

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

Set the size of the currently bound WebGLBuffer object.

void glBufferData(GLenum target, BufferDataSize size, const void* data, GLenum usage)

Set the size of the currently bound WebGLBuffer object.

If the passed data is null then an INVALID_VALUE error is generated.

void glBufferSubData(GLenum target, GLintptr offset, GLsizeiptr size, const void* data)

For the WebGLBuffer object bound to the passed target, update the buffer object an INVALID_VALUE error is generated. If the passed data is null then an INVALID_VALUE error is generated.

WebGLBuffer? createBuffer() ([OpenGL ES 2.0 §2.9](#), similar to [glCreateBuffer\(\)](#))

Create a WebGLBuffer object and initialize it with a buffer of size 0.

Figure 1:

2.9 Buffer Objects

The vertex data arrays described in section 2.8 are stored in client memory. It is sometimes desirable to store frequently used client data, such as vertex arrays, in high-performance server memory. GL buffer objects provide a mechanism for clients to use to allocate, initialize, and render from such memory.

The name space for buffer objects is the unsigned integers for the GL. A buffer object is created by binding an unused `GL_BUFFER`. The binding is effected by calling

```
void BindBuffer( enum target, uint buffer );
```

with *target* set to `ARRAY_BUFFER` and *buffer* set to the unused `GL_BUFFER`. The buffer object is a new state vector, initialized with a zero-sized array of data, comprising the state values listed in Table 2.5.

BindBuffer may also be used to bind an existing buffer object. If the binding is successful no change is made to the state of the newly bound buffer object. The previous binding to *target* is broken.

While a buffer object is bound, GL operations on the target `GL_BUFFER` affect the bound buffer object, and queries of the target to which the buffer is bound return state from the bound object.

Figure 2:

Example Implementation

As an example of implementation, we will see part of code in Firefox’s Browser Engine – Gecko.

- [WebGLRenderingContext.webidl](#)
- [RenderGL.h](#)
- [WebGLContextBuffers.cpp](#)

Conformity status of popular implementations

Older but more completed from [ref #1].

Desktop browsers

- Google Chrome – WebGL has been enabled on all platforms that have a capable graphics card with updated drivers since version 9, released in February 2011.
- Mozilla Firefox – WebGL has been enabled on all platforms that have a capable graphics card with updated drivers since version 4.0.
- Safari – Safari 6.0 and newer versions installed on OS X Mountain Lion, Mac OS X Lion and Safari 5.1 on Mac OS X Snow Leopard implemented support for WebGL, which was disabled by default before Safari 8.0.
- Opera – WebGL has been implemented in Opera 11 and 12, although was disabled by default in 2014.
- Internet Explorer – WebGL is partially supported in Internet Explorer 11.
- Microsoft Edge – The initial stable release supports WebGL version 0.95 (context name: “experimental-webgl”).

Mobile browsers

- BlackBerry 10 – WebGL is available for BlackBerry devices since OS version 10.00
- BlackBerry PlayBook – WebGL is available via WebWorks and browser in PlayBook OS 2.00
- Android Browser – Basically unsupported.
- Internet Explorer - WebGL is available on Windows Phone 8.1
- Firefox for mobile – WebGL is available for Android and MeeGo devices since Firefox 4.
- Firefox OS
- Google Chrome – WebGL is available for Android devices since Google Chrome 25 and enabled by default since version 30.
- Maemo – In Nokia N900, WebGL is available in the stock microB browser from the PR1.2 firmware update onwards.

- MeeGo - WebGL is unsupported in the stock browser “Web.” However, it is available through Firefox.
- Opera Mobile - Opera Mobile 12 supports WebGL (on Android only).
- Sailfish OS - WebGL is supported in the default Sailfish browser.
- Tizen - WebGL is supported
- Ubuntu Touch
- WebOS
- iOS – WebGL is available for mobile Safari, in iOS 8.

More updated information

You can check out the updated information in MDN [ref #2]

Support for WebGL is present in Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+ and Internet Explorer 11+; however, the user’s device must also have hardware that supports these features.

Online API documentation

The junior developer read books and tutorials; The senior developers read the API documentation; Only the platform developers and library writers read the standards and specs.

Why I should investigate into this part? The reason is simple: we want to know *how developer learns about the APIs and its intended usage*.

Thus, we should pay attention to the following several things:

1. How does the single interface documentation look like?
2. Can developer find the usage of some particular functionality quick?
3. Is there enough but concise examples along side?
4. How is the edge cases and correctness issues mentioned?
5. How is the compatibility, security and performance issues mentioned?

Sources

I found the MDN and MSDN have detailed API docs:

- [Mozilla Developer Network](#)
- [Microsoft Developer Network](#)

On the contrary, I don’t find API doc about WebGL on [Chromium’s online documentation](#);

[Dev.Opera](#) has a lot of tutorials but not API docs.

I find nothing significant on [Apple Developer site](#) either.

Interface structure

MDN

For example, [copyTexImage2D](#)

There are short induction, syntax, parameters and return value (semantics, types, ranges), examples, specification, browser compatibility, related interfaces.

MSDN

Same example, [copyTexImage2D](#)

Short intro, syntax, parameters and return value (semantics, more explicit types, ranges), remarks, WebGL errors, related interfaces

Usability

The navigation of MDN is better than MSDN. And most of time the Google will return MDN on searching a particular interface.

Also, the MDN provides per-interface examples and a lot of tutorials; MSDN is short at this.

Edge cases and correctness

Both consider edge cases and special values. The MSDN has a WebGL errors documentation, so it is more explicit about correctness.

Compatibility, security and performance

MDN

- Compatibility: Information about conformity in all major desktop and mobile browsers.
- Security: Not explicit
- Performance: Not explicit

MSDN

- Compatibility: Only one icon showing IE version supporting the interface.
- Security: Not explicit
- Performance: Not explicit

Ecosystem

Books

There are about 200 results returned for searching “WebGL” on amazon.com. As a comparison, searching “JavaScript” gives you 9000+ results.

The most popular books are:

WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL (OpenGL) by Kouichi Matsuda and Rodger Lea [ref #3]:

You’ll move from basic techniques such as rendering, animating, and texturing triangles, all the way to advanced techniques such as fogging, shadowing, shader switching, and displaying 3D models generated by Blender or other authoring tools. This book won’t just teach you WebGL best practices, it will give you a library of code to jumpstart your own projects.

Learning Three.js: The JavaScript 3D Library for WebGL by Jos Dirksen [ref #4]:

If you know JavaScript and want to start creating 3D graphics that run in any browser, this book is a great choice for you. You don’t need to know anything about math or WebGL; all that you need is general knowledge of JavaScript and HTML.

Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages by Tony Parisi [ref #5]:

In two parts—Foundations and Application Development Techniques—author Tony Parisi provides a thorough grounding in theory and practice for designing everything from a simple 3D product viewer to immersive games and interactive training systems. Ideal for developers with Javascript and HTML experience.

WebGL: Up and Running by Tony Parisi [ref #6]:

You don’t have to be a game development wizard or have 3D graphics experience to get started. If you use HTML, CSS, and JavaScript—and have familiarity with JQuery and Ajax—this book will help you gain a working knowledge of WebGL through clear and simple examples.

***WebGL Insights* by Patrick Cozzi [ref #7]:**

WebGL Insights shares experience-backed lessons learned by the WebGL community. It presents proven techniques that will be helpful to both intermediate and advanced WebGL developers.

It seems apparent that advanced materials share a high proportion. Also, one book is not really teaching WebGL but Three.js.

These popular books' publication dates range from 2012 to 2015; In all books, the newest is published in Feb 12, 2016 while the oldest is in Oct 5, 2011 as I found.

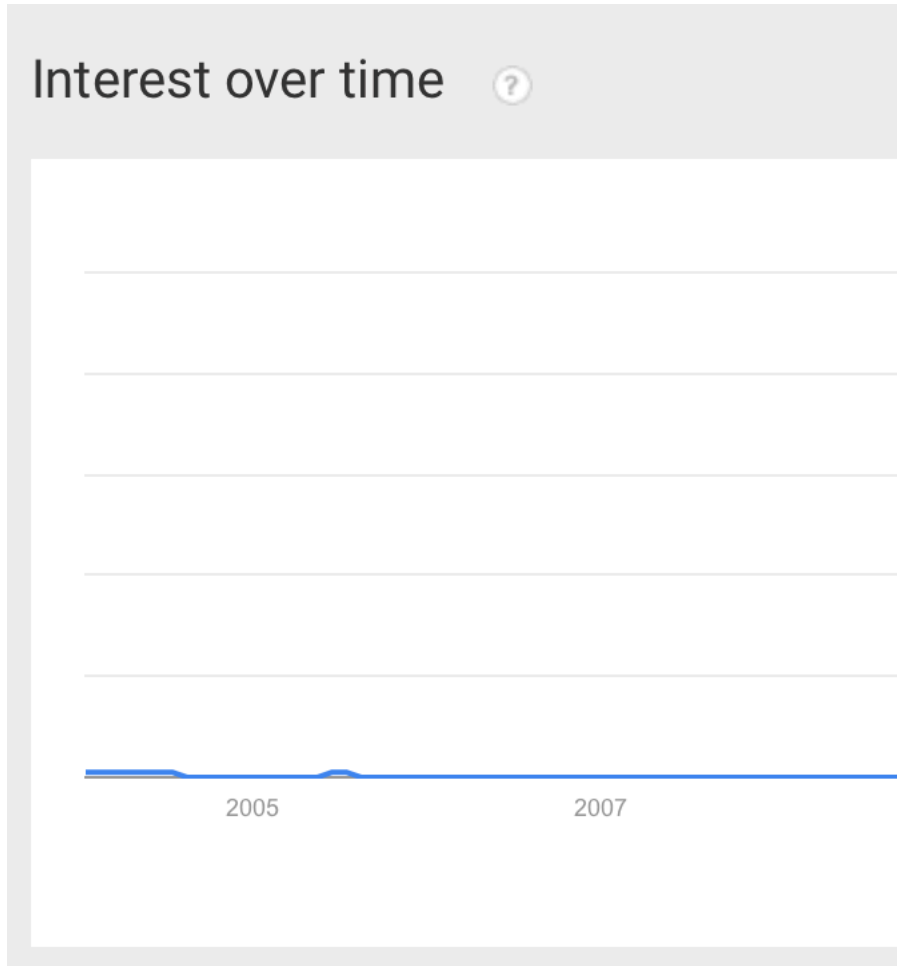
Tutorials

Google gives me about 373,000 results for “WebGL tutorial” (28,900,000 for “JavaScript” at the same time).

The popular ones are:

1. [MDN tutorial](#)
2. [LearningWebGL](#)
3. [WebGL Academy](#)
4. [WebGL Fundamentals](#)

Misc



- Google Trend for “WebGL”
- High-level libraries: BabylonJS, three.js, O3D, OSG.JS, CopperLicht and GLGE
- Game engines: Unreal Engine 4 and Unity 5
- Languages:
 - JavaScript (native)
 - TypeScript
 - PureScript
 - CoffeeScript
 - Haxe

Applications

Examples

- [NASA: Exploring Curiosity](#)
- [Chrome Experiments](#)
- [Three.js featured projects](#)

Chrome Experiments

Currently, it has 567 experiments using WebGL. (About half of all experiments).

From the first page, we will list the used technologies for each application (only ones include WebGL), “Open” means it is open-sourced.

1. [WebGL, Three.js, WebAudio; Open](#)
2. [Javascript, WebGL, Three.js, GLSL, WebRTC, WebAudio](#)
3. [Three.js, WebGL, Javascript](#)
4. [Three.js, Javascript; Open](#)
5. [Javascript es6, WebGL, Three.js, Glsl](#)
6. [Web Audio API, Tone.JS, pixi.JS, WebGL, WebRTC; Open](#)
7. [WebGL, WebAudio, Particulate.js, Three.js; Open](#)
8. [WebGL, Three.js, JavaScript \(ES2015\); Open](#)
9. [Javascript, WebGL, THREE.js, web audio API](#)

So as we can see, Three.js is really popular; And half of them are open sourced.

Error and warnings sampling

I tested the nine applications listed above, to see if there are any errors or warnings in my browser (The experiment UA: Safari 9). Here is the result for erros/warnings, and the rest is fine:

- From 2nd app: [Error] TypeError: undefined is not an object (evaluating 'camera.updateProjectionMatrix')
- From 3rd app: [Warning] THREE.Material: 'envMap' parameter is undefined. (three.min.js, line 428)
- From 4th app: [Warning] THREE.WebGLProgram: gl.getProgramInfoLog() - "WARNING: Output of vertex shader 'vNormal' not read by fragment shader " (three.js, line 29952)
- From 6th app: [Warning] THREE.WebGLProgram: gl.getProgramInfoLog() - "WARNING: Output of vertex shader 'vPosition' not read by fragment shader "

Part II

Troubles and Trouble-Shooting

Overview

Categories

In typical WebGL applications, we might have following kinds of errors (only the ones which *will* happen every time, we don't count the resource loading failure etc. in).

- GLSL
- Typo
- unexpected parameter
- Buffer not bound
- GLSL
- Resource deallocation
- Lack of certain step
- Buffer deallocation
- Type error
- Memory leak
- Improper sharing
- missing properties
- Name not in scope
- Undeclared identifier
- Context management
- Resource loading order
- API usage error
- Shader error
- Threejs
- Type errors
- Matrix related
- Others

GLSL errors

Referencing the GLSL errors in IE [ref #8], we can divide GLSL errors further into:

- Internal compiler error
- Compiler memory error - shader exceeds x bytes
- Syntax error - x
- Undeclared identifier x

- Invalid arguments passed to function `x`
- Postfix expression cannot be indexed
- Index out of range
- Incompatible index expression. For non-uniforms, the index must be an expression formed of the `loop_index` and integer constants. For uniforms, the index must be an integer constant.
- Index must be a constant
- Argument `x` is not a sampler
- Invalid macro name - cannot start with `GL_` or contain `__`
- Incompatible types in expression
- Expression in if statement does not evaluate to a boolean
- Divide or mod by zero in constant expression
- Invalid parameter count for macro
- Maximum uniform vector count exceeded
- Maximum attribute vector count exceeded
- Maximum varying vector count exceeded
- Maximum shader complexity exceeded
- Identifier already declared
- Invalid character used outside of comment
- Invalid initializer in for loop, needs to be a single variable of type float or int and initialized to a constant
- Invalid condition in for loop, needs to be in form `loop_index { > | >= | < | <= | == | != } constant`
- Invalid iteration in for loop, needs to be in form `{ --loop_index | ++loop_index | loop_index++ | loop_index-- | loop_index+=constant | loop_index-=constant }`
- Invalid modification to loop index inside loop body
- Invalid identifier name - cannot start with `gl_`, `webgl_`, `_webgl_` or contain `--`
- Token exceeds maximum length
- Invalid qualifier on array - cannot make arrays of attribute or const variables
- Incompatible type used for return expression
- Invalid qualifier on sampler variable declaration - must be uniform
- Invalid type passed to matrix constructor - arguments must be a matrix, or a scalar / vector of float / int / bool
- Invalid type passed to componentwise vector or matrix constructor - arguments must be a scalar / vector of float / int / bool
- Invalid argument count in componentwise vector or matrix constructor - total components passed must equal vector or matrix size
- Invalid expression on left of assignment expression
- Invalid swizzle in field selection - swizzle component count must be equal or less than max vector size (4)
- Invalid swizzle in field selection - swizzle components must be all from same set (xyzw, rgba or stpq)
- Invalid swizzle component in field selection - must be from a valid GLSL

set (xyzw, rgba or stpq)

- Swizzle component out of range - must select a component that exists in the vector
- This hardware is unable to support `gl_FrontFacing`
- Const variable requires initialization
- Variables declared with uniform, attribute, or varying qualifier cannot be initialized
- Varying variable cannot have bool, int, or struct type
- Invalid argument passed to constructor - argument must be a basic GLSL type
- Invalid type qualifier for function parameter - only const on in parameters is allowed
- Array declarator requires a constant expression
- Array was declared with size less than or equal to zero
- Type qualifiers `uniform` and `attribute` are invalid for structs
- Invalid field name for struct type
- Invalid type for left hand side of field selection
- Samplers are not allowed in structs
- Macros must be redefined the same as original definition
- Invalid loop index expression passed as out / inout parameter
- Type cannot be used as a constructor
- Undeclared type x
- Embedded struct declarations are not allowed
- Function x is declared and used but not defined
- Function redefinition not allowed
- Function redeclaration not allowed
- Invalid single argument to vector constructor - must be a scalar type, or another vector, or a 2x2 matrix
- Struct constructor arguments' types do not match the struct's field types
- Invalid location for continue statement - must be inside of a loop
- Cannot call main
- Invalid qualifier on non-global variable - non-global variables can be const but cannot be varying, attribute or invariant
- Cannot redefine main or define main with incorrect signature
- Cannot use reserved operators such as `~`, `%=`, `>>=`, `<<=`, `&=`, `|=`, or `^=`
- Ternary conditional operator must have boolean expression for test condition
- Ternary conditional operator must have two expressions of equal types after test condition
- Invalid location for break statement - break statements must be inside a loop
- Invalid location for discard statement - discard statements must be inside a fragment shader
- Initializer for const variable must initialize to a constant value
- Functions cannot be overloaded on return type

- Known functions cannot be re-declared or re-defined
- Function header definition parameter qualifiers must match declaration parameter qualifiers
- Array size must be an integer constant expression
- Array size expression too complex
- `#version` directive must specify 100 for version
- `#version` directive can only be preceded by whitespace or comments
- Unary operator not defined for type
- Struct declarations are disallowed in function parameter declarators
- Struct type declaration exceeds maximum nesting level of
- Operator not defined for struct types
- Operator not defined for user-defined types that contain array types
- Unknown extension: x
- Invalid behavior specified for extension - behavior must be require, warn, enable or disable for regular extensions, or warn or disable for `all`
- Required extension x is not supported
- Preprocessor directives can only be preceded by whitespace on a line
- Function declarations cannot be local
- Variable declared as type `void` is not allowed
- `'void'` is an invalid parameter type unless used as `(void)`

Compiled Information from MDN

Source: [API](#), [Best Practices](#)

Source of bugs

canvas element

To get the rendering context, i.e., the canvas element, we usually use `getElementById` or similar way to fetch the DOM object. The caveat is, the `id` is specified in HTML tag, such as

```
<canvas id="glcanvas" width="640" height="480">
```

And you must use the exact name in your JavaScript code. **It is possible that you will type it wrongly or refer to another unrelated canvas.**

gl Object

```
function initWebGL(canvas) {
    gl = null;

    try {
```

```

    // Try to grab the standard context. If it fails, fallback to experimental.
    gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
  }
  catch(e) {}

  // If we don't have a GL context, give up now
  if (!gl) {
    alert("Unable to initialize WebGL. Your browser may not support it.");
    gl = null;
  }

  return gl;
}

```

First, we might not have a functional `gl` (not `null`) always, which depends on the browser support. And second, in the `getContext`, we are also forced to consider two standards.

Shader's source

The shader code is essentially a string. So as a result, you might either define it in JavaScript, or store it as a HTML element, or even request it as a resource dynamically from other address.

```

if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
  alert("Unable to initialize the shader program.");
}

```

The above snippet checks if the `gl.linkProgram` calling succeeds by checking if return value is `null`. It is apparently something easy to forget.

Attribute names

The shader code will use some variable to communicate with JavaScript. For example

```

<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
  }
</script>

```

On the JavaScript side, we have to

```
// During Shader initialization
vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
gl.enableVertexAttribPointer(vertexPositionAttribute);

// During scene rendering
var pUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
gl.uniformMatrix4fv(pUniform, false, new Float32Array(perspectiveMatrix.flatten()));

var mvUniform = gl.getUniformLocation(shaderProgram, "uMVMMatrix");
gl.uniformMatrix4fv(mvUniform, false, new Float32Array(mvMatrix.flatten()));

gl.bindBuffer(gl.ARRAY_BUFFER, squareVerticesBuffer);
gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
```

So, we have to match the names (i.e. `aVertexPosition`, `uPMatrix` and `uMVMMatrix` in the above example) in two language domains, both in syntax and semantics.

Array

This is how to fill data into GL buffer: `gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);`

The `Uint16Array` is a raw, platform-dependent way of storing an array of data. Similarly, we also have `Array`, `Int8Array`, `Float32Array` ...

Interestingly, let compare it with OpenGL ES interface: `void BufferData(enum target, sizeiptr size, const void *data, enum usage);`.

You can see that, the `Uint16` array could be translated into a raw array and a element size indicator.

Things to avoid

- You should never use `#ifdef GL_ES` in your WebGL shaders; although some early examples used this, it's not necessary, since this condition is always true in WebGL shaders.
- Using **high** precision in fragment shaders will prevent your content from working on some older mobile hardware. You can use **medium** instead, but be aware that this often results in corrupted rendering due to lack of precision on most mobile devices, and the corruption is not going to be visible on a typical desktop computer. In general, only using **high** in both vertex and fragment shaders is safer unless shaders are thoroughly tested on a variety of platforms. Starting in Firefox 11, the WebGL `getShaderPrecisionFormat()` function is implemented, allowing you to

check if `high` precision is supported, and more generally letting you query the actual precision of all supported precision qualifiers.

- Anything that requires syncing the CPU and GPU sides is potentially very slow, so if possible you should try to avoid doing that in your main rendering loops. This includes the following WebGL calls: `getError()`, `readPixels()`, and `finish()`. WebGL getter calls such as `getParameter()` and `getUniformLocation()` should be considered slow too, so try to cache their results in a JavaScript variable.
- Simpler shaders perform better than complex ones. In particular, if you can remove an `if` statement from a shader, that will make it run faster. Division and math functions like `log()` should be considered expensive too.
- Always have vertex attrib 0 array enabled. If you draw with vertex attrib 0 array disabled, you will force the browser to do complicated emulation when running on desktop OpenGL (e.g. on Mac OS X). This is because in desktop OpenGL, nothing gets drawn if vertex attrib 0 is not array-enabled. You can use `bindAttribLocation()` to force a vertex attribute to use location 0, and use `enableVertexAttribArray()` to make it array-enabled.

Compiled Information from Stackoverflow

[Source](#)

Examples

Undeclared identifier

- [gl-color-is-undeclared-identifier-on-webgl](#)

Context management

- [another canvas.getContext\('2d'\) occupied this canvas context.](#)
- [webgl-scene-doesnt-render-because-of-lost-context](#)

Resource loading order

- [Try to render the scene before the shader program is downloaded and compiled](#)

API usage error

- [WebGL Fragment Shader constructor error - Too many arguments](#)
- [Where is your setup code](#)
- [WebGL error when attempting to get color data from vec4](#)
- [API name typo](#)
- [what-is-wrong-with-this-webgl-code](#)
- [Because `createProgram` don't return anything](#)
- [should be calling `gl.drawArrays\(\)` once for every object that is currently on the screen](#)
- [have to create an array buffer for each object you are drawing](#)
- [3 reasons of failing to call `gl.drawElements`](#)
- [webgl-drawarrays-with-invalid-mode-is-not-generating-a-error](#)
- [webgl-drawelements-out-of-range](#)
- [webgl-failing-at-drawing-points-gl-drawarrays-attempt-to-access-out-of-range](#)
- [webgl-invalid-value-attachshader-no-object-or-object-deleted-is-this-secret1](#)
- [why-doesnt-my-sphere-render-complete](#)
- [webgl-drawelements-out-of-range](#)
- [webgl-invalid-operation-useprogram](#)
- [type-canvasrenderingcontext2d-webglrenderingcontext-is-not-assignable-to-`typ`](#)
- [webgl-shader-errors](#)
- [what-will-happen-if-an-attribute-is-used-in-program-without-enabled-and-binding](#)
- [webgl-gl-error-gl-invalid-operation-gl-drawelements-attempt-to-access-out-of](#)

Shader error

- [uniform1i\(3, 0\) Is not valid WebGL](#)
- [GLSL interpreted as javascript](#)
- [WebGL - compileShader syntax error](#)
- [WebGL shader errors](#)
- [webgl-unable-to-initialize-shader-program](#)
- [google-chrome-webgl-shader-compile-linker-error-uniforms-with-the-same-name-but](#)
- [wrong-integer-math-in-webgl-shaders](#)
- [shader-compile-errors](#)
- [using-a-uniform-in-an-if-instruction-inside-a-fragment-shader-dont-work-since](#)
- [webgl-unable-to-initialize-shader-program](#)
- [gl-invalid-operation-caused-by-samplercube](#)

Threejs

- [Three.js: WebGL \(error\) drawing texture on a plane](#)
- [Normalize takes a vec3 not a vec4](#)
- [three.js-webgl-invalid-operation-bindtexture-object-not-from-this-context](#)
- [canvas-renderer-not-working](#)
- [three.js-particlesystem-creation-gives-invalid-operation-not-bound-buffer-array](#)

Others

- [WebGL: get error/warning message text as a string](#)
- [WebGL texture creation trouble](#)
- [Understanding WebGL State](#)
- [problems-with-texture-array-sending-to-shaders-in-webgl](#)
- [webgl-rendering-an-float32array-of-a-lot-of-elements-showing-out-of-range-vertic](#)
- [array-buffer-not-working-with-webgl](#)
- [webgl-texture-is-not-showing-correctly](#)
- [passing-color-to-fragment-shader-from-javascript](#)
- [is-there-a-lint-tool-for-opengl-shading-language](#)
- [three.js-shader-extension-errors](#)

Type errors

- [webgl-invalid-operation-vertexattribpointer-stride-or-offset-not-valid-for-ty](#)
- [webgl-glsl-shader-accessing-texture2d-overrides-other-texture](#)
- [So finally it is a typing mistake](#)

Matrix related

- [webgl-using-gl-matrix-library-mat4-translate-not-running](#)
- [square-doesnt-appear-using-perspective-matrix](#)

Compiled Information from three.js

Overview

- [threejs-issue-5421: Buffer deallocation](#)
- [threejs-issue-5569: Type mismatch](#)
- [threejs-issue-5680: Memory leak](#)

- threejs-issue-5871: Improper sharing
- threejs-issues-83: Name not in scope
- threejs-pull-1602: GLSL
- threejs-issue-4834: Unexpected parameter
- threejs-issue-5098: Buffer not bound
- threejs-issue-5196: Buffer not bound
- threejs-issue-5222: GLSL & Unexpected parameter
- threejs-issue-5269: Resource deallocation
- threejs-issue-5293: Lack of certain step
- threejs-issue-6952: Missing properties
- threejs-issue-6956: GLSL

Details

threejs-pull-1602

Categories

- GLSL

Link

<https://github.com/mrdoob/three.js/pull/1602>

Remark

This PR mentioned a problem about adding precision qualifiers to resolve shader compilation errors on mobile device. This is also a nasty problem which is platform-dependent ### threejs-issue-1329 #### Categories - Typo

Link

<https://github.com/mrdoob/three.js/issues/1329>

Remark

`THREE.UnsignedIntType` is written wrongly as `THREE.UnsignedShortType`. However, its namespace is `THREE`, which is defined by library, not by WebGL. #### Possible fix I think TAJIS might be able to resolve this by analyzing information of object property? ### threejs-issue-4834 #### Categories - unexpected parameter

Link

<https://github.com/mrdoob/three.js/issues/4834>

Remark

Discussed 2 problems here related to the compilation failure of the shader.

- [This commit](#) used `max()` to limit negative values for `pow()` in shaders.
- Then there is a discussion about the warning X3557 caused by `MAX_DIR_LIGHTS` is 1 and the redundant loop:

```
#define MAX_DIR_LIGHTS 1
#if MAX_DIR_LIGHTS > 0
for( int i = 0; i < MAX_DIR_LIGHTS; i++ ) { ...
```

Possible fix

maybe same as [threejs issue 5222](#) ### threejs-issue-5098 #### Categories - Buffer not bound

Link

<https://github.com/mrdoob/three.js/issues/5098>

Remark

An eye caught error.

`THREE.Float32Attribute` has been removed.

Use `THREE.BufferAttribute(array, itemSize)` instead.

this [commit](#) solves this:

```
- geometry.addAttribute( 'position', new THREE.Float32Attribute( numEdges * 2 * 3, 3 ) );
+ geometry.addAttribute( 'position', new THREE.BufferAttribute( new Float32Array( numEdges * 2
```

Possible fix

Search for available functions in the lib ### threejs-issue-5196 #### Categories - Buffer not bound

Link

<https://github.com/mrdoob/three.js/issues/5196#issue-40214568>

Remark

Found while playing with threejs.org/editor

After adding a Mesh with the Menubar's 'Add' panel, sidebar geometry parameter changes cause the object to stop displaying.

Console warnings from WebGL: WebGL: INVALID_OPERATION: vertexAttribPointer: no bound ARRAY_BUFFER three.min.js:545 WebGL: INVALID_OPERATION: vertexAttribPointer: no bound ARRAY_BUFFER three.min.js:549 WebGL: INVALID_OPERATION: drawElements: no ELEMENT_ARRAY_BUFFER bound three.min.js:552 [17:54:37] Saved state to IndexedDB. 0.71ms Storage.js:64 ##### Possible fix Track the gl context and the buffer might avoid these. ### threejs-issue-5222 ##### Categories - GLSL - unexpected parameter

Link

<https://github.com/mrdoob/three.js/issues/5222>

Remark

As referred to to [GLSL manual P65](#), in `genType pow (genType x, genType y)`, results are undefined if $x = 0$ and $y \leq 0$. So in the generated GLSL in the issue, `pow(pointDotNormalHalf, shininess)` could be undefined and not warned. In the commit [#7252](#), this was fixed by: `- uniforms.shininess.value = material.shininess; + uniforms.shininess.value = Math.max(material.shininess, 1e-4);` ##### Possible fix Interval analysis and warning ##### Related - [messed up OBJMTL loader #7252](#) - [MeshPhongMaterial with shininess = 0 causes artifacts on Windows #6057](#)

threejs-issue-5269

Categories

- Resource deallocation

Link

<https://github.com/mrdoob/three.js/issues/5269>

Remark

The child of object3D was not removed before removing the object3D itself. This arouse the memory leakage. ##### Possible fix Might be fixed by tracking the state of the object? ### threejs-issue-5293 ##### Categories - Lack of certain step

Link

<https://github.com/mrdoob/three.js/issues/5293>

Remark

WebGLRenderer is not doing `updateObject(object)` because is not visible from the main camera. hen an error `glDrawElements: range out of bounds for buffer` would occur. ##### Possible fix Might be fixed by tracking the state of the object? ##### Related - The author broke it again recently in [#6996](#)

threejs-issue-5421

Categories

- Buffer deallocation

Link

<https://github.com/mrdoob/three.js/issues/5421>

Remark

According to the [commit](#), the problem is that we should not only call `_gl.deleteBuffer(buffer_obj)`, but also call `delete buffer_obj` [Document about deleteBuffer'](https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/deleteBu##### Possible fix The problem might cause the old state to persist unexpectedly. So one way of elimination is to require a “fresh” state explicitly at some point, which can preclude such presence if possible. ##### threejs-issue-5569##### Categories - Type error

Link

<https://github.com/mrdoob/three.js/commit/1311c0e315326bdb9c02a5c7b8733bb0c27fb1ea>

Remark

[Commit](#). Well, it is recognised by naked eyes. But first it is a silent bug, which is rather harmful. Second, although threejs doesn't depend on `gl-matrix.js`, it has a similar internal system ##### Possible fix Add additional type information ##### threejs-issue-5680 ##### Categories - Memory leak

Link

<https://github.com/mrdoob/three.js/issues/5680>

Remark

Fixed in [PR](#), with [commit](#). This is very nasty ... I can't give a reasonable fix now. ##### threejs-issue-5871 ##### Categories - Improper sharing

Link

<https://github.com/mrdoob/three.js/issues/5871>

Remark

This is caused by the geometry being shared across contexts outside of the closure in **ArrowHelper**'s definition. [A length related discussion](#). ##### Possible fix Globally, we might be able to couple a context with a buffer. ##### threejs-issue-6952 ##### Categories - missing properties

Link

<https://github.com/mrdoob/three.js/pull/6952>

Remark

When exporting **BoxGeometry** parameters(width, height, depth) by **SceneExporter**, it should be **g.parameters.width** rather than **g.width**. Fixed by this [commit](#). ##### Possible fix Track object's properties. ##### Related - SceneExporter doesn't handle box/sphere/plane parameters #4739 - SceneExporter // BoxGeometry Inconsistent #5067 - #5067 make SceneExporter use correct BoxGeometry parameters #5068

threejs-issue-6956

Categories

- GLSL

Link

<https://github.com/mrdoob/three.js/issues/6956>

Remark

Pull: <https://github.com/thothbot/parallax/pull/43> In GLSL, extension directive must occur before any non-preprocessor tokens, otherwise a warning is raised. Threejs fixed in 72dev. ##### Related - [parallax](#) - [ancient-earth](#)

threejs-issues-83

Categories

- Name not in scope

Link

<https://github.com/mrdoob/three.js/issues/83>

Remark

The `scene` referenced isn't passed in as an argument or declared above - so it's looking in the global scope. So it tends to be some programming style issue, which is related to the problem being solved.

threejs-pull-1602**Categories**

- GLSL

Link

<https://github.com/mrdoob/three.js/pull/1602>

Remark

This PR mentioned a problem about adding precision qualifiers to resolve shader compilation errors on mobile device. This is also a nasty problem which is platform-dependent

Compiled Information from applications

NOTE: needs more feedback from community

fireworks-webgl

[Repo](#)

Issues:

- [drawing buffer alert](#)

medusae

[Repo](#)

Issues:

- [Fix alpha material shaders](#)

Trouble-shooting WebGL code

Professional WebGL Programming: Developing 3D Graphics for the Web

A list of problems:

- JavaScript syntax error
- Runtime error
- Compilation error in the shader
- Linking error in the shader program
- If your fragment shader tries to use a varying variable that is not defined in your vertex shader
- WebGL specific errors

Trouble-shooting checklist (only debug the code part):

1. Check that you didn't misspell a name of an object property.
2. Check you have spelled all properties of `WebGLRenderingContext` correctly.

An Introduction to WebGL Programming

[Link](#)

Just as with regular programs, a syntax error from the compilation stage, or a missing symbol from the linker stage could prevent the successful generation of an executable program. There are routines for verifying the results of the compilation and link stages of the compilation process, but are not shown here. Instead, we've provided a routine that makes this process much simpler, as demonstrated on the next slide.

Beginning WebGL for HTML5

Chapter 9: Debugging and Performance

The main error codes are:

- `INVALID_ENUM`
- `INVALID_VALUE`
- `INVALID_OPERATION`
- `OUT_OF_MEMORY`

Context errors:

1. Context creation — might fail to obtain a WebGL context

Misc

- <http://www.gamedev.net/topic/673408-debugging-pure-webgl-and-js-is-hell/>
- <https://yulian.kuncheff.com/using-intellijwebstorm-to-debug-web-applications/>

Statistics

Collected from the bugs in three.js sampling and stackoverflow sampling.

- GLSL: 4
- Typo: 1
- Unexpected parameter: 2
- Buffer not bound: 2
- Resource deallocation: 1
- Lack of certain step: 1
- Buffer deallocation: 1
- Memory leak: 1
- Improper sharing: 1
- missing properties: 1
- Name not in scope: 1
- Undeclared identifier: 1
- Context management: 2
- Resource loading order: 1
- API usage error: 20
- Shader error: 11
- Threejs: 5
- Type errors: 4
- Matrix related: 2
- Others: 10

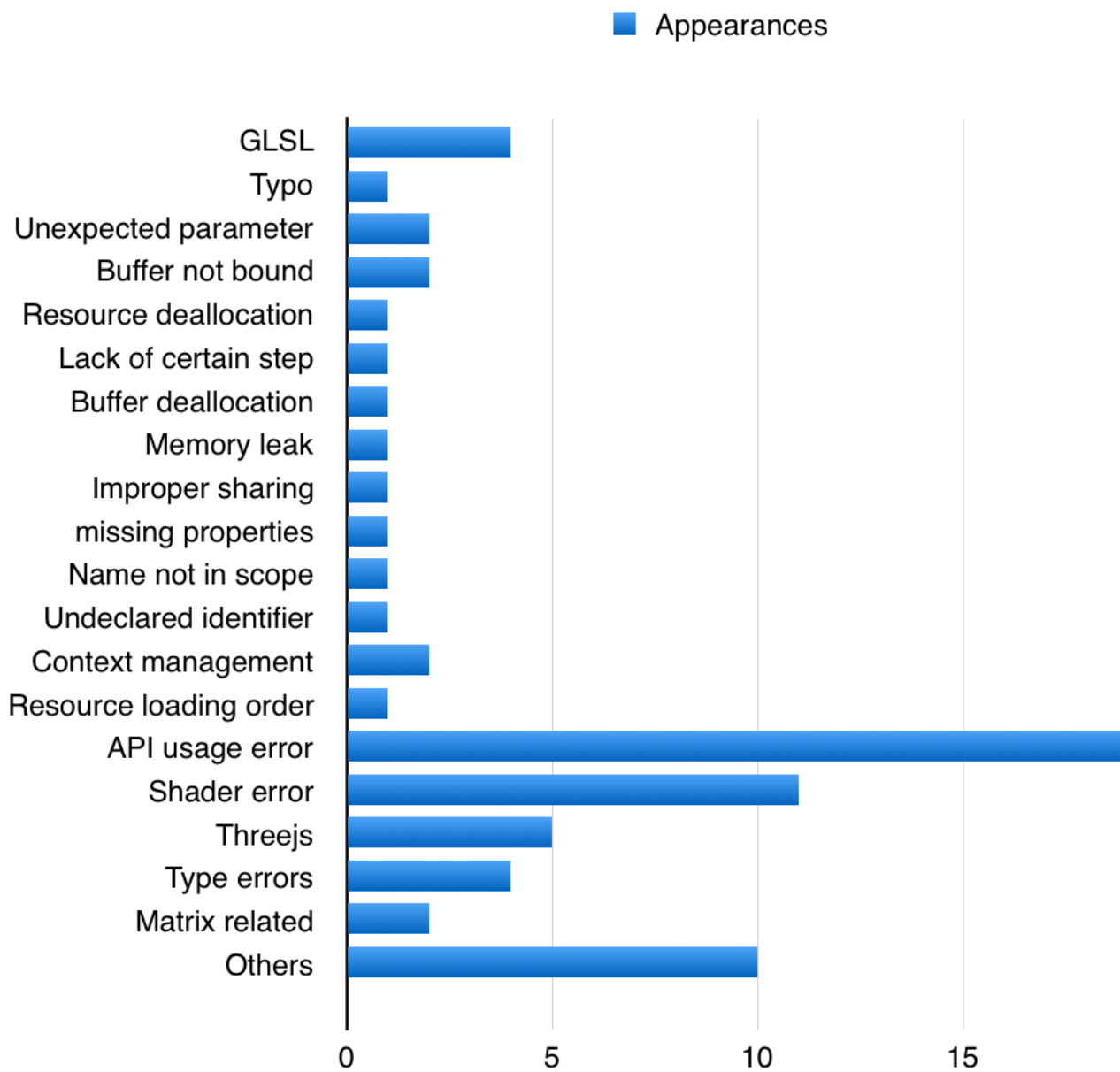


Figure 3:

Part III

Library Support

Overview

Popular libraries

- High-level libraries: BabylonJS, three.js, O3D, OSG.JS, CopperLicht and GLGE
- Matrix libraries: gl-matrix, sylvester

Abstractions

Take three.js as an example:

- Geometry
- Objects
- Materials
- Lights
- Resource loaders
- Linear algebra
- Scenes
- Camera
- Renderer (not just WebGL)
- Textures

Why libraries?

1. The developer can focus on the logic of their applications/business, rather than the implementation details;
2. The third-party library's APIs are better designed to fit the language construct of JavaScript
3. The high-level library is simpler and more restricted in effects – which means less error-prone.

As a result, the following errors can be eliminated by using a library:

- Typo
- Buffer binding
- Resource Management
- Lack of certain step
- Missing properties
- Name not in scope

- Undeclared identifier
- API usage error
- Matrix related

Case study: gl-matrix

[Home](#)

glMatrix is designed to perform vector and matrix operations stupidly fast! The latest version uses WebPack to manage the modules.

APIs

Exposed interfaces: `glMatrix`, `mat2`, `mat2d`, `mat3`, `mat4`, `quat`, `vec2`, `vec3`, `vec4`

Common utilities

3x3 Matrix

Library structure

- `glMatrix`: Common utilities, including config constants, compatibility detection, and things like `setMatrixArrayType`.
- `mat3` etc: Represent one type of data and the related operations.
 - Constructors: `create`, `clone`, `copy`
 - Computations: `identity`, `transpose`
 - Conversions: `fromMat4`

Possible problems

1. **Hard to maintain**: Even such a simple library is over 6000 lines of JS; and the author suggested a “sorry” for breaking the APIs from 1.0 to 2.0.
2. **Type Unsafe**: Concretely, it is easy to pass a `vec3` to where a `vec4` is expected. And since here is no type checking, so you won’t get any explicit warning.

Case Study: Threejs

API samples

[Docs](#)

Class glMatrix

Common utilities

Defined in: [common.js](#).

Methods

`glMatrix.setMatrixArrayType(type)`

Sets the type of array used when creating new vectors and matrices

Method Detail

`glMatrix.setMatrixArrayType(type)`

Sets the type of array used when creating new vectors and matrices

Parameters:

`{Type} type`

Array type, such as `Float32Array` or `Array`

Figure 4:

Class mat3

3x3 Matrix

Defined in: [mat3.js](#).

Methods

`mat3.adjoint(out, a)`

Calculates the adjugate of a mat3

`mat3.clone(a)`

Creates a new mat3 initialized with values from an existing matrix

`mat3.copy(out, a)`

Copy the values from one mat3 to another

Figure 5:

Method Detail

{mat3} mat3.adjoint(out, a)

Calculates the adjugate of a mat3

Parameters:

{mat3} out

the receiving matrix

{mat3} a

the source matrix

Returns:

{mat3} out

Figure 6:

A example app, its workflow:

- initialize `scene`, `camera` and `renderer`.
- Create mesh object from geometry and material add them to the scene
- Render based on scene and camera frame by frame

```
<html>
  <head>
    <title>My first Three.js app</title>
    <style>
      body { margin: 0; }
      canvas { width: 100%; height: 100% }
    </style>
  </head>
  <body>
    <script src="js/three.min.js"></script>
    <script>
      var scene = new THREE.Scene();
      var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 100 );

      var renderer = new THREE.WebGLRenderer();
      renderer.setSize( window.innerWidth, window.innerHeight );
      document.body.appendChild( renderer.domElement );

      var geometry = new THREE.BoxGeometry( 1, 1, 1 );
      var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
      var cube = new THREE.Mesh( geometry, material );
      scene.add( cube );

      camera.position.z = 5;

      var render = function () {
        requestAnimationFrame( render );

        cube.rotation.x += 0.1;
        cube.rotation.y += 0.1;

        renderer.render(scene, camera);
      };

      render();
    </script>
  </body>
</html>
```

Library structure

The current three.js implementation is too huge – I will take its early release `three.js-r16` as a preliminary analysis object.

The `src` contains

1. cameras: set up the perspective matrix etc. based on camera parameters
2. core
 1. Color: convert hexadecimal code into internal representation and better format
 2. Face3/Face4: Wrap end-points and normal vector into a high-level structure `Face`
 3. Geometry: A geometry is a set of vertices and faces connecting the vertices, this function computes the normals of each face (which might be useful in texture or fragment shader?)
 4. Vector(2, 3, 4)/Matrix4: Basically same as `gl-matrix`
 5. Vertex: Wrapper over position and normal (normal?)
 6. UV: (u, v) coordinate (NOTE: UV mapping is a process of flattening the 3-dimensional object)
 7. Rectangle: Again, a geometry wrapper
3. materials: Wrap the attribute values of different materials
4. objects: Simple wrappers of `Line`, `Mesh`, `Object3D` etc.
5. renderers
 1. `renderables`: `RenderableFace(3,4)`, `RenderableLine`, `RenderableParticle` etc., looks like another set of wrappers
 2. `CanvasRenderer`: In construction, context is get from created `canvas` element; A lot of other vectors and rectangle are created as well. The functions provided include `setSize`, `clear`, `render`, `drawTexturedTriangle` and `expand`. However, this is only a 2d canvas.
 3. `Renderer`: Its data includes pools of face3, face4, line and particles, as well as a vector4 and a matrix4. The exposed interfaces include a `renderList` and method `project`. I suppose that this will do some transformation (like projection), and push the things left to render into `renderList`
 4. `SVGRenderer`: Similar to `CanvasRenderer`
 5. `WebGLRenderer`: Similar to `CanvasRenderer`, we have some basic bootstrapping and after that, we call `initGL` and `initProgram`; The utilities provided include `setSize`, `clear`, `render`, `getShader`; There are also internal functions `getShader` and `matrix2Array`.
 - `initGL`: It will try to get context, and will throw in case of incompatibility. If context is ready, it will do `clearColor`, `clearDepth` and other config and setups.
 - `initProgram`: Two shaders, fragment and vertex shaders are hard-coded here. With attached shaders, it link and use the program. Finally, it will set other attributes related.

- **clear:** Clear the `COLOR_BUFFER_BIT` and `DEPTH_BUFFER_BIT` bits.
 - **render:** Given a `scene` and a `camera`, it will render the mesh object in `scene` one by one. Every mesh object will have its vertex buffer. The related data also contains faces and color. For every face, its three vertices will be push into the array, same for color. With buffers ready, it will create, bind and fill in one by one with `createBuffer`, `bindBuffer` and `bufferData`. After that, the view matrix and projection matrix is set. Next, the material of object is rendered as well. The color of face are pushed into the buffer, bind and filled in. Finally, we will call `drawElements`.
 - **getShader:** It is basically wrapping around `createShader`, `shaderSource`, `compileShader`.
6. `scenes`: Wrapper of an array of objects

Part IV

Tool Support

Exclusive Debug tools

Some debugging tools for WebGL development are listed here.

[Another post](#)

WebGL Insights

[Home](#)

This tool looks very powerful and mature. Look at the features:

- Chrome Extension embedded in the Chrome DevTools panel
- Overdraw Inspector
- Mipmap Inspector
- Depth Inspector
- Call Stack Timeline and Statistics
- Program Usage Count
- Duplicate Program Usage Detector
- Program Viewer
- Frame Control
- State Variable Editor
- Resource Viewer

One screen-shot:

WebGL-Inspector

[Home](#)

Features

- Extension for injecting into pages
- Embed in an existing application with a single script include
- Capture entire GL frames
- Annotated call log with stepping/resource navigation and redundant call warnings
- Pixel history - see all draw calls that contributed to a pixel + blending information
- GL state display
- Resource browsers for textures, buffers, and programs



Developer Tools - http://webglsamples.c



Elements

Console

Network

Shaders

Usages

Programs

State

Profiles

Call Stack 0

Recent Draw: Call Stack 0

Recent Draw: Call Stack 1

Histogram 0

Quake 3 WebGL Demo

media.tojicode.com/q3bsp/

captured frame 34090

FPS: 29

Trace

Timeline

State

Textures

Buffers

Programs

3991

5046

30923

31799

34090

depthMask(true)

clear(256)

bindBuffer(ELEMENT_ARRAY_BUFFER, [Buffer 12])

bindBuffer(ARRAY_BUFFER, [Buffer 11])

disable(CULL_FACE)

blendFunc(ONE, ZERO)

depthMask(false)

depthFunc(LEQUAL)

useProgram([Program 25])

activeTexture(TEXTURE0)

uniform1i("texture", 0)

bindTexture(TEXTURE_2D, [Texture 93])

[Demo](#)

This is a very convenient debugger, it can be useful when you application's logic is wrong (compared to our analyzer's target – the syntax and API is wrong).

webgl-debug.js

- [Repo](#)
- [Home](#)

It will wrap the `WebGLRenderingContext` with a debugging wrapper, which will make any GL errors show up in the JavaScript console of browser.

Also, we can log function calls with by passing a logger in the above mentioned wrapper.

It provides [a sample](#).

WebGL Linter

[Repo](#)

This is a discontinued and very primitive project – but the idea is there.

Browser Support and IDE Support

Browser

Chrome Canvas Inspector

See [the post](#)

Firefox Shader Editor

See [this post](#)

IDE

Generally speaking, by using an IDE, you can avoid a lot of silly bugs all-together.

Also, there are some graphical editors for modeling.

JetBrain - WebStorm

It has [support](#) for GLSL language by plugin.

WebGL Studio

[Home](#)

Three.js editor

[Home](#)

3Dmol.js

[Home](#)

Blend4Web

[Home](#)

Part V

Conclusion

Best practices in WebGL development

General ones

1. If you are writing code – Use a good IDE or modern editor, such as WebStorm, Eclipse, and Atom. This can help you prevent most silly errors when you are still a newbie.
2. Use the graphical editor to get your model (in fact, you can model in softwares like Blender and Maya, then import the model into the JS).
3. Make error explicit – Check return value always, catch possible errors.
4. Take advantages of debugging tools like WebGL inspector.

Technical ones

- [Article from MDN](#)
- [glQuery/webgl-best-practices](#)

Future of WebGL tooling

Static analysis technique

The interoperation between JavaScript and GLSL opens a door for static analysis; Besides, editors and current IDE is unaware of if you have `check null`; The dynamic type information is simply ignored.

Also, the correct use of API involves side-effects, which is easy to get wrong.

Verified libraries

Three.js is a huge code base, and highly dynamic. You never know if some feature does work or it is just coincidence.

By verifying:

1. The library does what it is intended to do
2. The library does its job the right way

We will have more confidence and rely on it in the long run.

Integration

The integration into mature product is important, such as debug tools and IDEs.