WebGL is a software interface for accessing graphics hardware from within a web browser. Based on OpenGL ES 2.0, WebGL allows a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of 3D objects.

- [n.n.n] refers to sections in the WebGL 1.0 specification, available at www.khronos.org/webgl
- Content marked in purple does not have a corresponding function in OpenGL ES. The OpenGL ES 2.0 specification is available at www.khronos.org/registry/gles

WebGL function calls behave identically to their OpenGL ES counterparts unless otherwise noted.

The WebGL Context and getContext() [2.5]

This object manages OpenGL state and renders to a drawing buffer, which must also be created at the same time as the context creation. Create the WebGLRenderingContext object and a drawing buffer by calling the getCanvas method of a given HTMLCanvasElement object with the exact string 'webgl'. The drawing buffer is also created by getCanvas.

Interfaces

Interfaces are optional requests and may be ignored by an implementation. See getCanvasAttributes for actual values.

WebGLContextAttributes [5.2]

This interface contains requested drawing surface attributes and is passed as the second parameter to getCanvas.

Attributes:

- **alpha**: Default: true
  - If true, requests a drawing buffer with an alpha channel for the purposes of performing OpenGL destination alpha operations and compositing with the page.

- **depth**: Default: true
  - If true, requests a drawing buffer with a depth buffer of at least 16 bits.

- **stencil**: Default: false
  - If true, requests a stencil buffer of at least 8 bits.

- **antialias**: Default: true
  - If true, requests drawing buffer with antialiasing using its choice of technique (multisample/supersample) and quality.

- **premultipliedAlpha**: Default: true
  - If true, requests drawing buffer which contains colors with premultiplied alpha. (Ignored if Alpha is false.)

- **preserveDrawingBuffer**: Default: false
  - If true, requests that contents of the drawing buffer remain in before frames, at potential performance cost.

Per-Fragment Operations [5.13.3]

- **void blendColor(float red, float green, float blue, float alpha)**
- **void blendEquation(enum mode)**
- **void blendEquationSeparate(enum modeRGB, enum modeAlpha)**
- **void blendFuncSeparate(enum sfactor, enum dfactor)**
- **void clearDepth(float depth)**
- **void clearColor(float red, float green, float blue, float alpha)**
- **void clearStencil(int stencil)**
- **void colorMask(bool red, bool green, bool blue, bool alpha)**
- **void depthMask(bool mask)**
- **void stencilMask(int mask)**
- **void stencilMaskSeparate(enum face, int mask)**

ArrayBuffer and Typed Arrays [5.12]

- **Data is transferred to WebGL using ArrayBuffer and views. Buffers represent unstructured binary data, which can be modified using one or more typed array views. Buffers**

Buffers

- **ArrayBuffer(along byteLength)**
  - byteLength: read-only, length of view in bytes.
  - Creates a new buffer. To modify the data, create one or more views referencing it.

- **View**
  - In the following, ViewType may be Int8Array, Int16Array, Int32Array, Uint8Array, Uint16Array, Uint32Array, Float32Array, or ArrayBuffer.
  - ViewType(along length)
  - Creates a view and a new underlying buffer. viewLength: Read-only, number of elements in this view.
  - ViewType(ViewType other)
  - Creates new underlying buffer and copies 'other' array.
  - ViewType(viewType other)
  - Creates new underlying buffer and copies 'other' array.

Whole Framebuffer Operations [5.13.3]

- **void clear(along mask)**
  - mask: bitwise OR of [COLOR, DEPTH, STENCIL, BUFFER_BITMAP]
  - clearColor(float red, green, blue, alpha)
  - clearDepth(float depth)

Buffer Objects [5.5]

- **Once bound, buffers may not be rebound with a different Target.**
- **void bindBuffer(target, Object buffer)**
- **void createBuffer(Object target)**
  - Note: Corresponding OpenGL ES function is GenBuffers

View and Clip [5.13.3 - 5.13.4]

- **The viewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Drawing buffer size is determined by the HTMLCanvasElement.**
- **void depthRange(float zNear, float zFar)**
  - zNear: Clamped to the range 0 to 1
  - zFar: Clamped to the range 0 to 1
- **void scissor(int x, int y, long width, long height)**
- **void viewport(int x, int y, long width, long height)**
Programs and Shaders [5.13.9]

Rendering with OpenGL ES 2.0 requires the use of shaders. Shaders must be loaded with a source string (shaderSource), compiled (compileShader), and attached to a program (attachShader) which must be linked (linkProgram) and then used (useProgram).

```plaintext
void attachShader(Object program, Object shader)
void bindAttribLocation(Object program, uint index, string name)
void compileShader(Object shader)
void detachShader(Object program, Object shader)
Object createProgram()
Object createShader(enum type)
  type: VERTEX_SHADER, FRAGMENT_SHADER
void deleteProgram(Object program)
Object getAttachedShaders(Object program)
any getProgramParameter(Object program, enum pname)
  Note: Corresponding OpenGL ES function is GetProgramiv
  pname: DELETE_STATUS, LINK_STATUS, VALIDATE_STATUS, ATTACHED_SHADERS, ACTIVE_ATTRIBUTES, (UNIFORMS)
string getProgramInfoLog(Object program)
any getShaderParameter(Object program, enum pname, int index)
  Note: Corresponding OpenGL ES function is GetShaderiv
  Argument type: SHADER_TYPE, DELETE_STATUS, (UNIFORMS)
string getShaderInfoLog(Object program)
any getShaderParameter(Object program, enum pname)
  Note: Corresponding OpenGL ES function is GetShaderiv
  Argument type: (SHADER_TYPE, (ATTRIBUTE_LOCATION))
string getShaderSource(Object shader)
blob isShader(Object program, Object shader)
void linkProgram(Object program)
void shaderSource(Object shader, string source)
void validateProgram(Object program)
```

Textures Objects [5.13.8]

Texture objects provide storage and state for texturing operations. WebGL adds an error for operations relating to the currently bound texture if no texture is bound.

```plaintext
void activeTexture(enum texture)
  texture: TEXTURE_0, TEXTURE_CUBE_MAP

void copyTexImage2D(enum target, int level, GLenum internalFormat, int x, int y, long width, long height, int border)
  target: TEXTURE_2D, TEXTURE_CUBE_MAP

void copyTexSubImage2D(enum target, int level, int xoffset, int yoffset, long width, long height)
  target: See target for copyTexImage2D

Object createTexture() 
Note: Corresponding OpenGL ES function is glGenTextures
```

Uniforms and Attributes [5.13.10]

Values used by the shaders are passed in as uniform of vertex attributes.

```plaintext
void disableVertexAttribArray(uint index)
  Argument index: [0, MAX_VERTEX_ATTRIBS - 1]
void enableVertexAttribArray(uint index)
  Argument index: [0, MAX_VERTEX_ATTRIBS - 1]
Object getActiveAttrib(Object program, uint index)
Object getActiveUniform(Object program, uint index)
ulong getUniformLocation(Object program, string name)
any getProgramParameter(Object program, enum pname)
  Note: Corresponding OpenGL ES function is GetProgramiv
  Argument pname: DEACTIVATE_ALL_ATTRIBS, (UNIFORMS)
string getProgramInfoLog(Object program)
any getShaderParameter(Object program, enum pname, int index)
  Note: Corresponding OpenGL ES function is GetShaderiv
  Argument type: (SHADER_TYPE, (ATTRIBUTE_LOCATION))
string getShaderInfoLog(Object program)
any getShaderParameter(Object program, enum pname)
  Note: Corresponding OpenGL ES function is GetShaderiv
  Argument type: (SHADER_TYPE, (ATTRIBUTE_LOCATION))
string getShaderSource(Object shader)
blob isShader(Object program, Object shader)
void linkProgram(Object program)
void shaderSource(Object shader, string source)
void validateProgram(Object program)
```

Framebuffer Objects [5.13.6]

Framebuffer objects provide an alternative rendering target to the drawing buffer.

```plaintext
void bindFramebuffer(enum target, Object framebuffer)
target: FRAMEBUFFER
enum checkFramebufferStatus(enum target)
target: FRAMEBUFFER
Returns: FRAMEBUFFER_COMPLETE, (UNCOMPLETE), FRAMEBUFFER_INCOMPLETE_ATTACHMENT, (ATTACHMENT_DIMENSIONS,MISSING_ATTACHMENT)
Object createFramebuffer() 
Note: Corresponding OpenGL ES function is glGenFramebuffers
void deleteFramebuffer(Object buffer)
```

Special Functions [5.13.3]

```plaintext
context Struct getContextAttribs() 
void disable(enum cap)
  cap: BLENDFUNC, (STENCIL) 
void enable(enum cap)
  cap: See cap for disable
void finish()
void flush()
```

eval [5.13.11]

```plaintext
return: SUCCESS, (NOMEMORY), INVALID_OPERATION, FRAMEBUFFER_OPERATION, VALUE, NO_ERROR, CONTEXT_LOST, WEBGL
```
Types [4.1]

A shader can aggregate these using arrays and structures to build more complex types. There are no pointer types.

Basic Types
- void: no function return value or empty parameter list
- bool: Boolean
- int: signed integer
- float: floating scalar
- vec2, vec3, vec4: n-component floating point vector
- bvec2, bvec3, bvec4: Boolean vector
- mat2, mat3, mat4: 2x2, 3x3, 4x4 float matrix
- sampler2D: access a 2D texture
- samplerCube: access cube mapped texture
- struct-name: members
- [ ]; struct-name: members
- coordinates
- normals
- texture coordinates

Structures and Arrays [4.1.8, 4.1.9]

- Structures
  - struct type-name (members)
  - struct-name();
  - // optional variable declaration,
  - // optionally an array

- Arrays
  - float foo[];
  - *: structures and blocks can be arrays
  - *: only 1-dimensional arrays supported
  - *: structure members can be arrays

Operators and Expressions [5.1]

Numbered in order of precedence. The relational and equality operators < < <= != == = evaluate to a Boolean. To compare structures component-wise, use functions such as lessThan(), equal(), etc.

Operators
- ( )
- [ ]
- : (selects one entire operand. Use (m) to select individual components of vectors.)
- *= /= += -=
- * / + - !
- & | ^
- <= >= != == =

Preprocessor [3.4]

Preprocessor Directives

The number sign (#) can be immediately preceded or followed in its line by spaces or horizontal tabs.

Examples of Preprocessor Directives
- "#version 1.0" in a shader program specifies that the program is written in GLSL ES version 1.00. It is optional. If used, it must occur before anything else in the program other than whitespace or comments.
- "extension extension_name;": behavior, where behavior can be require, enable, warn, or disable; and where extension_name is the extension supported by the compiler.

Predefined Macros

| __LINE__ | LINE integer that is one more than the number of preceding new-lines in the current source string |
| __VERSION__ | VERSION integer, e.g.: 100 |
| GL_ES | Defined and set to integer 1 if running on an OpenGL ES Shading Language |
| GL_FRAGMENT_PRECISION_HIGH | 1 if highp is supported in the fragment language, undefined |

Qualifiers

Storage Qualifiers [4.3]

Variable declarations may be preceded by one storage qualifier.

- none
- const: Compile-time constant, or read-only function parameter
- uniform: Value does not change across the primitive being processed, uniform form the linkage between a shader, OpenGL ES, and the application
- varying: Linkage between a vertex shader and fragment shader for interpolated data

Uniform [4.3.4]

Use to declare global variables whose values are the same across the entire primitive being processed. All uniform variables are read-only. Use uniform qualifiers with any basic data types, to declare a variable whose type is a structure, or an array of any of these. For example:

```glsl
uniform vec3 lightPosition;
```

Varying [4.3.5]

The varying qualifier can be used only with the data types float, vec2, vec3, vec4, mat2, mat3, mat4, or arrays of these. Structures cannot be varying. Varying variables are required to have global scope. Declaration is as follows:

```glsl
varying vec3 normal;
```

Parameter Qualifiers [4.4]

Input values are copied in at function call time, output values are copied out at function return time.

- none [Default] local read/write memory, or input parameter
- in: For function parameters passed into a function
- inout: For function parameters passed back out of a function, but not initialized for use when passed in
- out: For function parameters passed both into and out of a function

Precision and Precision Qualifiers [4.5]

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

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

For example:
- lowp float color;
- varying mediump vec2 Coord;
- lowp vec2 foo(lowp mat3);
- highp mat4 m;

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

<table>
<thead>
<tr>
<th>FP Range</th>
<th>FP Magnitude Range</th>
<th>FP Precision</th>
<th>Integer Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>highp</td>
<td>(−214, 214)</td>
<td>Relative 2−14</td>
<td>(−214, 214)</td>
</tr>
<tr>
<td>mediump</td>
<td>(−214, 214)</td>
<td>Relative 2−14</td>
<td>(−214, 214)</td>
</tr>
<tr>
<td>lowp</td>
<td>(−2, 2)</td>
<td>Absolute 2−1</td>
<td>(−2, 2)</td>
</tr>
</tbody>
</table>

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

Invariant Qualifiers Examples [4.6]

```glsl
extension extension_name
```}

<table>
<thead>
<tr>
<th>#pragma GLSL Invariant(all)</th>
<th>Force all output variables to be invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Qualify a previously declared variable</td>
</tr>
<tr>
<td>invariant varying mediump vec3 Color;</td>
<td>Qualify as part of a variable declaration</td>
</tr>
</tbody>
</table>

Order of Qualification [4.7]

When multiple qualifications are present, they must follow a strict order. This order is as follows: invariable, storage, precision, parameter, precision

Aggregate Operations and Constructors [5.4]

Matrix Constructor Examples [5.4]

```glsl
mat2(float, float, float, float);
mat2(vec2, vec2);
mat2(float, float, float, float);
mat2(vec2, float, float, float);
```

Structure Constructor Example [5.4.3]

```glsl
struct light { int intensity; float position;);
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

Matrix Components [5.6]

Access components of a matrix with an array subscript syntax. For example:

```glsl
m = m +/- m;
```

Structure Operations [5.7]

Select structure fields using the period (.) operator. Other operators include:

```glsl
field selector
:= = assignment
```

Array Operations [4.1.9]

Array elements are accessed using the array subscript operator [] for example:

```glsl
diffuseColor += lightIntensity[3] * NdotL;
```
Built-In Functions

Component-wise operation. Parameters specified as gl_MaxDrawBuffers.

Outputs:
- mediump vec4 gl_FragCoord; fragment position within frame buffer
- mediump vec2 gl_PointCoord; fragment position within a point (point rasterization only)

Fragment Shader Special Variables [7.2]
Fragment shaders may write to gl_FragColor or to one or more elements of gl_FragData[].

Built-In Functions

Angle & Trigonometry Functions [8.1]
These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

Vector Relational Functions [8.6]
These functions compare x and y component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is bvec [4].

Matrix Functions [8.5]
Type mat t is any matrix type.

Geometric Functions [8.4]
These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

Exponential Functions [8.2]
Component-wise operation. T is float, vec2, vec3, vec4.

Common Functions [8.3]
Component-wise operation. T is float, vec2, vec3, vec4.

Texture Lookup Functions [8.7]
Available only in fragment shaders.

Statements and Structure

Iteration and Jumps [6]

Sample Program

A shader pair that applies diffuse and ambient lighting to a textured object.

Built-In Constants With Minimum Values [7.4]

These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

WebGL and OpenGL ES are registered trademarks of Khronos Group. The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See www.khronos.org to learn about the Khronos Group. See www.khronos.org/webgl to learn about WebGL. See www.khronos.org/opengles to learn about OpenGL ES.