WebGL™ is an immediate-mode 3D rendering API designed for the web. It is derived from OpenGL® ES 3.0, and provides similar rendering functionality, but in an HTML context. WebGL 2 is not entirely backwards compatible with WebGL 1. Existing error-free content written against the core WebGL 1 specification, with or without extensions will often run in WebGL 2 without modification, but this is not always the case.

The WebGL 2 specification shows differences from the WebGL 1 specification. Both WebGL specifications are available at khr.org/webgl. Unless otherwise specified, the behavior of each method is defined by the OpenGL ES 3.0 specification. The OpenGL ES specification is at khr.os/registry/gles/.

### Interfaces

**WebGLContextAttributes [5.2]**

This interface contains requested drawing surface attributes and is passed as the second parameter to `getContext`. Some of these are optional requests and may be ignored by an implementation.

- `alpha` (Default: true)
  - If true, requests a drawing buffer with an alpha channel for the purposes of performing OpenGL constant_alpha and compositing with the painting.

- `depth` (Default: true)
  - If true, requests drawing buffer with a depth buffer of at least 16 bits. Must obey.

- `stencil` (Default: false)
  - If true, requests a stencil buffer of at least 8 bits. Must obey.

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

- `premultipliedAlpha` (Default: false)
  - 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 between frames, at potential performance cost. May have significant performance implications on some hardware.

- `preferLowPowerToHighPerformance` (Default: false)
  - Provides a hint suggesting that implementation create a context that optimizes for power consumption over performance.

- `failIfMajorPerformanceCaveat` (Default: false)
  - If true, context creation will fail if the performance of the created WebGL context would be dramatically lower than that of a native application making equivalent OpenGL calls.

**WebGLObject [5.3]**

This is the parent interface for all WebGL resource objects:

- `WebGLBuffer [5.4]`
  - Created as if by `genBuffers`, bound by `bindBuffer`, destroyed by `deleteBuffers` in OpenGL ES.

- `WebGLFramebuffer [5.5]`
  - Created as if by `genFramebuffers`, bound by `bindFramebuffer`, destroyed by `deleteFramebuffers` in OpenGL ES.

- `WebGLProgram [5.6]`
  - Created as if by `createProgram`, used by `useProgram`, destroyed by `deleteProgram` in OpenGL ES.

- `WebGLRenderbuffer [5.7]`
  - Created as if by `genRenderbuffers`, bound by `bindRenderbuffer`, destroyed by `deleteRenderbuffers` in OpenGL ES.

- `WebGLShader [5.8]`
  - Created as if by `createShader`, attached to program by `attachShader`, destroyed by `deleteShader` in OpenGL ES.

- `WebGLTexture [5.9]`
  - Created as if by `genTextures`, bound by `bindTexture`, destroyed in OpenGL ES.

- `WebGLUniformLocation [5.10]`
  - Location of a uniform variable in a shader program.

- `WebGLActiveInfo [5.11]`
  - Information returned from calls to `getActiveAttrib` and `getActiveUniform`. The read-only attributes are:

    - `size`
    - `type`
    - `name`
    - `precision`

- `WebGLShaderPrecisionFormat [5.12]`
  - Information returned from calls to `getShaderPrecisionFormat`. The read-only attributes are:

    - `maxLod`
    - `maxLevel`
    - `maxPrecision`

**WebGLQuery [3.2]**

- `WebGLQuery [3.3]`
  - Created as if by `genQueries`, made active by `beginQuery`, destroyed by `deleteQueries` in OpenGL ES.

- `WebGLSync [3.4]`
  - Created as if by `fenceSync`, blocked on by `waitSync`, queried by `getSync`, destroyed by `deleteSync` in OpenGL ES.

**WebGLBuffer [5.4]**

- `bufferData`: Creates a new buffer. To modify the data, create one or more typed array views. Consult the `ArrayBuffer` and `ArrayBufferView` specification for more details on Typed Arrays.

### Per-Fragment Operations [5.13.4]

- `void blendColor(byte red, byte green, byte blue, byte alpha)`

- `void blendEquation([enum mode])`
  - `mode`: See `blendRGB` for `blendEquationSeparate`

- `void blendEquationSeparate([enum modeRGB], [enum modeAlpha])`

- `void blendFunc([enum sfactor], [enum dfactor])`
  - `sfactor` and `dfactor` may not both reference constant color

- `void blendFuncSeparate([enum srcRGB], [enum dstRGB], [enum srcAlpha], [enum dstAlpha])`
  - `srcRGB`, `scAlpha`: See `blendfunc` for `blendfuncSeparate`

- `void colorMask([bool r, bool g, bool b, bool a])`

- `void combination([enum sfactor], [enum dstRGB], [enum dstAlpha], [enum function])`

- `void depthMask([bool enable])`

- `void depthRanging([bool enable], [float zNear], [float zFar])`

- `void stencilMask([bool enable], [enum stencilBuffer])`

- `void readPixels([int x], [int y], [int width], [int height], [enum format], [enum type], [int dstOffset])`

- `void drawBuffers([enum drawBuffers])`

- `void copyBuffers([enum copyBuffers])`

### ArrayBuffer and Typed Arrays [5.13.3]

- `Data is transferred to WebGL using ArrayBuffers and views. Buffers represent unstructured binary data, which can be modified using one or more typed array views. Consult the ECMA-262 specification for more details on Typed Arrays.

**Buffer [5.13.4]**

- `ArrayBuffer [5.13.4]`
  - `bufferLength`: length-only, read-only, length of view in bytes.
  - `bufferCopy`: creates a new buffer. To modify the data, create one or more views referencing it.

**Views [5.13.5]**

- `Int8Array`, `Int16Array`, `Int32Array`, `Uint8ClampedArray`, `Uint8Array`, `Uint16Array`, `Uint32Array`, `Float32Array`, `Float64Array`:
  - Creates a new view and a new underlying buffer.
  - `length`: read-only, number of elements in this view.

**Typed Arrays [5.13.6]**

- `new Int8Array()`:
  - Creates a new underlying buffer and copies other array.

**Methods [5.13.7]**

- `view[i] = setElement i`

- `set(viewType other, [optional length])`

**Other Properties [5.13.8]**

- `byteLength`: read-only, length of view in bytes.
- `byteOffset`: length-only, element-size in bytes.

**Buffer Objects [5.13.4]**

- `void bufferData([enum target], [BufferDataSource data], [optional usage])`
  - target and usage: Same as for bufferData above

- `void bufferSubData([enum target], [targetOffset], [sourceOffset])`
  - target: target for bindBuffer

- `void copyBufferSubData([enum target], [buffer], [bufferOffset], [target], [targetOffset], [size])`

- `getBufferSubData([enum target], [target], [source], [targetOffset], [size])`
Buffer Objects (continued)

Object createBuffer();

Corresponding OpenGL ES function is GenBuffers;

do deleteBuffer(WebGLBuffer? buffer);

google any getBufferParameter(enum target, enum pname);

target: See target for bindBuffer

pname: BUFFER_SIZE, BUFFER_USAGE

bool isBuffer(WebGLBuffer? buffer);

Detect and Enable Extensions [5.14][5.14.9][5.3.7] .

string [ getSupportedExtensions()];

object getExtension(string name);

Available in the WebGLRenderingContext interface.

Get information about the context

callContextGetExtensionAttributes();

Set and get state

Calls in this group behave identically to their OpenGL ES counterparts unless otherwise noted. Source and destination factors may not both reference constant color.

Programs and Shaders [5.14.9][5.3.7].

Shaders are loaded with a source string (shaderSource), compiled (compileShader), attached to a program (attachShader), linked (linkProgram), then used (useProgram).

[WebGLHandlesContextLost](int getFragmentDataLocation(WebGLProgram program, DOMString name);

attachShader(Object program, Object shader);

void detachShader(Object program, Object shader);

Object [ getAttachedShaders(Object program, Object);] any getProgramParameter(WebGLProgram? program, enum pname);

Corresponding OpenGL ES function is GetProgramiv

pname: DELETE_STATUS, LINK_STATUS, VALIDATE_STATUS, ATTACHED_SHADERS, ACTIVE_ATTRIBUTES, UNIFORMS, ACTIVE_UNIFORM_BLOCKS, TRANSFORM_FEEDBACK_BUFFER_MODE, TRANSFORM_FEEDBACK_VARYINGS

string getProgramInfoLog(Object program);

any getShaderParameter(Object shader, enum name);

Corresponding OpenGL ES function is GetShaderiv

name: SHADER_TYPE, DELETE_STATUS, COMPARE_STATUS

string getShaderInfoLog(Object shader);

getShaderSource(Object shader);

bool isProgram(Object program);

bool isShader(Object shader);

void linkProgram(Object program);

void shaderSource(Object shader, string source);

void useProgram(Object program);

void validateProgram(Object program);

Special Functions [5.13.3][5.3.7.2].

contextStruct containsGetContextAttributes();

void disable(enum cap);

Vertex Array Objects [3.7.17]
VAOs encapsulate all state related to the definition of data used by the vertex processor.
void bindVertexArray(WebGLVertexArrayObject? vertexArray);
WebGLVertexArrayObject? createVertexArray();
void deleteVertexArray(WebGLVertexArrayObject vertexArray);
[WebGLHandlesContextLost] boolean isVertexArray(WebGLVertexArrayObject vertexArray);

Texture Objects [5.14.8] [3.7.6]
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.
void activeTexture(enum target) [5.14.3] texture: TEXTURE_TEXTURE_2D] where i = MAX_BOUND_TEXTURE_IMAGE_UNITS - 1
void bindTexture(enum target, WebGLTexture? texture); target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP
void copyTexImage2D(enum target, int level, enum internalformat, int x, int y, int, long, long, int, border);
void copyTexSubImage2D(enum target, int, int, int, int, x, int, int, int, x, int, long, long, int, border);
void texImage2D(target, xoffset, yoffset, source, width, height, format, type, texImage3D(source, target, internalformat, width, height, format, type, mipMap=True, srcOffset=0, dstOffset=0, level=0, border=0)
void texImage3D(target, int, int, int, xoffset, yoffset, zoffset, source, width, height, depth, format, type, texImage3D(source, target, internalformat, width, height, depth, format, type, level=0, border=0)

Framebuffer Objects [5.14.6] [3.7.4]
Framebuffer objects provide an alternative rendering target to the drawing buffer.
void bindFramebuffer(enum target, WebGLFramebuffer? framebuffer); target: READ_FRAMEBUFFER
[WebGLHandlesContextLost] enum checkFramebufferStatus(webglError); target: READ_FRAMEBUFFER
Returns: FRAMEBUFFER_COMPLETE [COMPLETE, UNSUPPORTED], FRAMEBUFFER_INCOMPLETE_ATTACHMENT [INCOMPLETE_ATTACHMENT, DIMENSIONS, MULTISAMPLE, MISSING_ATTACHMENT], FRAMEBUFFER_UNDEFINED
Object createFramebuffer();
Corresponding OpenGL ES function is GenFramebuffers
void deleteFramebuffer(Object buffer);
void framebufferTexture2D(enum target, attachment, enum textureTarget, enum texture, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP
void framebufferRenderbuffer(target, attachment, enum renderbufferTarget, WebGLRenderbuffer? renderbuffer); target: RENDERBUFFER
attachment: COLOR_ATTACHMENT0...COLOR_ATTACHMENT16
renderbufferTarget: RENDERBUFFER
bool isFramebuffer(WebGLFramebuffer framebuffer);
void framebufferTextureLayer(target, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, level: 0...LEVELS - 1
void blitFramebuffer(target, x0, y0, x1, y1, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, source, srcX0, srcY0, srcX1, srcY1, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, dstX0, dstY0, dstX1, dstY1, field mask, filter);
void framebufferTextureLayer(target, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, level: 0...LEVELS - 1
void invalidateFramebuffer(target, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, source, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, source, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP
void readBuffer(target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP

Renderbuffer Objects [5.14.7] [3.7.5]
Renderbuffer objects are used to provide storage for the individual buffers used in a framebuffer object.
void bindRenderbuffer(target, Object renderbuffer); target: RENDERBUFFER Object createRenderbuffer();
Corresponding OpenGL ES function is GenRenderbuffers
void deleteRenderbuffer(Object renderbuffer);
any getRenderbufferParameter(target, enum renderbufferParameterName, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP
void getRenderbufferStorage(target, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, width, height, internalFormat, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP
void getRenderbufferStorageMultisample(target, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP, target: TEXTURE_2D, 3D, 2_ARRAY], TEXTURE_CUBE_MAP
Whole Framebuffer Operations [5.14.3]
void clear(bitrfield mask);
  mask: bitwise OR of [COLOR, DEPTH, STENCIL, BUFFER_BIT]
void clearColor(clamp red, clamp green, clamp blue, clamp alpha);
void clearDepth(float depth);
  depth: Clamped to the range 0 to 1.
void clearStencil(int s);
void colorMask(bool red, bool green, bool blue, bool alpha);
void depthMask(bool flag);
void stencilMask(uint mask);
void stencilMaskSeparate(uint face, uint mask);
  face: FRONT, BACK, FRONT_AND_BACK

Multiple Render Targets [3.7.11]
void drawBuffers(sequence<GLenum> buffers);
void clearColorBufferi(enum buffer, int drawbuffer, Int32List values, uint srcOffset=0);
void clearColorBufferi(enum buffer, drawbuffer, Int32List values, uint srcOffset=0);
void clearColorBufferi(enum buffer, drawbuffer, uint32List values, uint srcOffset=0);
void clearColorBufferi(enum buffer, drawbuffer, float depth, int stencil);

Sampler Objects [3.7.13]
WebGLSampler? createSampler();
void deleteSampler(WebGLSampler? sampler);
[WebGLHandlesContextLoss] boolean isSampler(WebGLSampler? sampler);
void bindSampler(uint unit, WebGLSampler? sampler);
void samplerParameterf(WebGLSampler sampler, enum pname, float param);
void samplerParameteri(WebGLSampler sampler, enum pname, int param);

Query Objects [3.7.12]
WebGLQuery? createQuery();
void deleteQuery(WebGLQuery? query);
[WebGLHandlesContextLoss] boolean isQuery(WebGLQuery? query);
void beginQuery(enum target, WebGLQuery query);
void endQuery(enum target);
WebGLQuery? getQuery(enum target, enum pname);
target: ANY_SAMPLES_PASSED, ANY_SAMPLES_PASSED_CONSERVATIVE,
TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN
pname: CURRENT_QUERY,
any getQueryParameter(WebGLQuery query, enum pname);
  pname: QUERY_RESULT_AVAILABLE

Sync Objects [3.7.14]
Synchronize execution between the GL server and the client.
WebGLSync? fenceSync(enum condition, bitfield flags);
[WebGLHandlesContextLoss] boolean isSync(WebGLSync? sync);
void deleteSync(WebGLSync? sync);
enum clientWaitSync(WebGLSync sync, bitfield flags, uint64 timeout);
  flags: SYNC_FLUSH_COMMANDS_BIT
void waitSync(WebGLSync sync, bitfield flags, int64 timeout);
  timeout: TIMEOUT_IGNORED
any getSyncParameter(WebGLSync sync, enum pname);
  pname: OBJECT_TYPE, SYNC_CONDITION, Flags, STATUS

Uniform Buffer Objects [3.7.16]
Provides the storage for named uniform blocks.
void bindBufferBase(enum target, uint index, WebGLBuffer? buffer);
void bindBufferRange(enum target, uint index, WebGLBuffer? buffer, intptr offset, sizeiptr size);
sequence<DOMString> getUniformIndices(WebGLProgram program, sequence<DOMString> uniformNames);
any getActiveUniforms(WebGLProgram program, ...
uniformNames, enum pname);
any getActiveUniformBlockParameter(WebGLProgram program, ...
uniformBlockIndex, enum pname);

OpenGL Texture Object and Sampler State
### Sized Texture Color Formats [3.7.11]

If an application wants to store the texture at a certain resolution or in a certain format, it can request the resolution and format with internalFormat. The following table shows the sized internal formats indicating whether they are color renderable or texture filterable.

In **Color Renderable** column, a red Y means the aiff extension EXT_color_buffer_float is enabled. In **Texture Filterable** column, a red Y means the iff extension OES_texture_float_linear is enabled.

<table>
<thead>
<tr>
<th>Internal Format</th>
<th>Format</th>
<th>Type</th>
<th>Color Renderable</th>
<th>Texture Filterable</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8</td>
<td>RED</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>R8_SNORM</td>
<td>RED</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R16F</td>
<td>RED</td>
<td>HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>R32F</td>
<td>RED</td>
<td>FLOAT</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>R8UI</td>
<td>RED_INTEGER</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R8I</td>
<td>RED_INTEGER</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R16UI</td>
<td>RED_INTEGER</td>
<td>UNSIGNED_SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R16I</td>
<td>RED_INTEGER</td>
<td>SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R32UI</td>
<td>RED_INTEGER</td>
<td>UNSIGNED_INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R32I</td>
<td>RED_INTEGER</td>
<td>INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB</td>
<td>RG</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB_SNORM</td>
<td>RG</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RG16F</td>
<td>RG</td>
<td>HALF_FLOAT,FLOAT</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RG32F</td>
<td>RG</td>
<td>FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8UI</td>
<td>RG_INTEGER</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8I</td>
<td>RG_INTEGER</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB16UI</td>
<td>RG_INTEGER</td>
<td>UNSIGNED_SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB16I</td>
<td>RG_INTEGER</td>
<td>SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32UI</td>
<td>RG_INTEGER</td>
<td>UNSIGNED_INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32I</td>
<td>RG_INTEGER</td>
<td>INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SRGB8</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA8</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGBA8_SNORM</td>
<td>RGBA</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8_565</td>
<td>RGB</td>
<td>UNSIGNED_BYTE, UNSIGNED_SHORT_5_6_5</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGBA8_SNORM</td>
<td>RGB</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>R11F_G11F_B10F</td>
<td>RGB</td>
<td>UNSIGNED_INT_10F_11F_11F_REV, HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB9_E5</td>
<td>RGB</td>
<td>UNSIGNED_INT_5_9_9_9_REV, HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB8_16F</td>
<td>RGB</td>
<td>HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8_32F</td>
<td>RGB</td>
<td>FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8UI</td>
<td>RGB_INTEGER</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8I</td>
<td>RGB_INTEGER</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB16UI</td>
<td>RGB_INTEGER</td>
<td>UNSIGNED_SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB16I</td>
<td>RGB_INTEGER</td>
<td>SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32UI</td>
<td>RGB_INTEGER</td>
<td>UNSIGNED_INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32I</td>
<td>RGB_INTEGER</td>
<td>INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA8</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SRGB8_ALPHA8</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA8_SNORM</td>
<td>RGBA</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA5_A1</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE, UNSIGNED_SHORT_5_5_5_1, UNSIGNED_INT_2_10_10_10_REV</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGBA4</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE, UNSIGNED_SHORT_4_4_4_4</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGBA10_A2</td>
<td>RGBA</td>
<td>UNSIGNED_INT_2_10_10_10_REV</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGBA16F</td>
<td>RGBA</td>
<td>HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA32F</td>
<td>RGBA</td>
<td>FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA8UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8A1I</td>
<td>RGBA_INTEGER</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA10_A2UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_INT_2_10_10_10_REV</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA16UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA16I</td>
<td>RGBA_INTEGER</td>
<td>SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA32I</td>
<td>RGBA_INTEGER</td>
<td>INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBA32UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_INT</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
The OpenGl ES Shading Language is two closely-related languages which are used to create shaders for the vertex and fragment processors contained in the WebGL, OpenGL, and OpenGL ES processing pipelines. WebGL 2.0 is based on OpenGL ES 3.0.

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, uint signed, unsigned integer
- float floating scalar
- vec2, vec3, vec4 n-component floating point vector
- bvec2, bvec3, bvec4 Boolean vector
- ivec2, ivec3, ivec4 signed integer vector
- uvec2, uvec3, uvec4 unsigned integer vector
- mat2, mat3, mat4 2x2, 3x3, 4x4 float matrix
- mat2x2, mat2x3, mat2x4 2x2, 2x3, 2x4 float matrix
- mat3x2, mat3x3, mat3x4 3x2, 3x3, 3x4 float matrix
- mat4x2, mat4x3, mat4x4 4x2, 4x3, 4x4 float matrix

Floating Point Sampler Types (opaque)
- sampler2D, sampler3D access a 2D or 3D texture
- samplerCube access cube mapped texture
- samplerCubeShadow access cube map depth texture with comparison
- sampler2DShadow access 2D depth texture with comparison
- sampler2DArray access 2D array texture
- sampler2DArrayShadow access 2D array depth texture with comparison

Signed Integer Sampler Types (opaque)
- isampler2D, isampler3D access an integer 2D or 3D texture
- iscSamplerCube access integer cube mapped texture
- iscSampler2DArray access integer 2D array texture

Unsigned Integer Sampler Types (opaque)
- usampler2D, usampler3D access unsigned integer 2D or 3D texture
- uscSamplerCube access unsigned integer cube mapped texture
- uscSampler2DArray access unsigned integer 2D array texture

Structures and Arrays [4.1.8, 4.1.9]
- struct struct-name { members struct-name[ ] // optional variable declaration, // optionally an array
- Arrays float foo[3]; Structures, blocks, and structure members can be arrays. Only 1-dimensional arrays supported.

Qualifiers [4.3]
Storage Qualifiers may be preceded by one storage qualifier.

- none (Default) local read/write memory, or input parameter
- const Compile-time constant, or read-only function parameter
- in Linkage into a shader from a previous stage
- out Linkage out of a shader to a subsequent stage
- in/out Linkage into a shader from a previous stage and linkage out of a shader to a subsequent stage
- uniform Value does not change across the primitive being processed, uniforms form the linkage between a shader, OpenGL ES, and the application

The following interpolation qualifiers for shader outputs and inputs may precede in, centroid in, out, or centroid out.

- smooth Perspective correct interpolation
- flat No-interpolation

Interface Blocks [4.3.7]
Uniform variable declarations can be grouped into named interface blocks, for example:

```glsl
uniform Transform {
  mat4 ModelViewProjectionMatrix;
  uniform mat3 NormalMatrix; // restatement of qualifier float Deformation;
};
```

Layout Qualifiers [4.3.8]
Layout qualifier identifiers for uniform blocks, for example:

```glsl
layout(layout-qualifier) block-declaration
```

Input Layout Qualifiers [4.3.8.1]
For all shader stages:

- location = integer-constant

Output Layout Qualifiers [4.3.8.2]
For all shader stages:

- location = integer-constant

Uniform Block Layout Qualifiers [4.3.8.3]
Layout qualifier identifiers for uniform blocks:
- shared, packed, std140, {row, column}_major

Order of Qualification [4.7]
When multiple qualifications are present, they must follow a strict order. This order is either:

- invariant, interpolation, storage, precision
- or:
- storage, parameter, precision

Preprocessor [3.4]
Preprocessor Directives
The number sign (#) can be immediately preceded or followed in its line by spaces or horizontal tabs.

- #define
- #undef
- #if
- #ifdef
- #else
- #endif
- #extension
- #line

Examples of Preprocessor Directives
- "#extension version 3.00" must appear in the first line of a shader program written in GLSL ES version 3.00. If omitted, the shader will be treated as targeting version 1.00.
- "#extension extension-name : behavior, where behavior can be require, enable, warn, or disabled; and where extension-name is the extension supported by the compiler.
- "#pragma optimize(on, off)" - enable or disable shader optimization (default on)
- "#pragma debug(on, off)" - enable or disable compiling shaders with debug information (default off)

Predefined Macros
- _LINE_ Decimal integer constant that is one more than the number of preceding newline in the current source string
- _FILE_ Decimal integer constant that says which source string number is currently being processed.
- _VERSION_ Decimal integer, e.g.: 300
- GL_ES Defined and set to 1 if running on an OpenGL-ES Shading Language.

Parameter Qualifiers [4.4]
Input values are copied at call function time, output values are copied out at function return time.

- none (Default) same as in
- in For function parameters passed into a function
- out For function parameters passed back out of a function, but not initialized for use when passed in
- inout 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.
- mediump Range and precision is between that provided by lowp and highp.
- lowp Range and precision can be less than mediump, but still represents an all color values for any color channel.

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

- highp (-20, 20], 0.0, 2-126, 2127)
- mediump (-21, 21], 2-10, 215)
- lowp (-2, 2], 2-3, Absolute 2(-2, 2-1]

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]
- #pragma STDGL invariant(all) Forc e all output variables to be invariant
- invariant gl_Position; Qualify a previously declared variable
- invariant centroid out vec3 Color; Qualify as part of a variable declaration

Operators [5.1]
Numbered in order of precedence. The relational and equality operators > < <= == != evaluate to a Boolean. To compare vectors-component-wise, use functions such as lessThan(), equal(), etc. [8.7.7].

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td>relational</td>
<td>L &gt; R</td>
</tr>
<tr>
<td>==</td>
<td>equality</td>
<td>L = R</td>
</tr>
<tr>
<td>&amp;</td>
<td>bit-wise and</td>
<td>L &amp; R</td>
</tr>
<tr>
<td>^</td>
<td>bit-wise exclusive or</td>
<td>L ^ R</td>
</tr>
<tr>
<td></td>
<td>bit-wise inclusive or</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>selection (Selects an entire operand. Use mix! to select individual components of vectors.)</td>
<td>L</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
<td>L = R</td>
</tr>
<tr>
<td>%</td>
<td>arithmetic assignments</td>
<td>L % R</td>
</tr>
<tr>
<td>+, -</td>
<td>additive</td>
<td>L</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>bit-wise shift</td>
<td>L</td>
</tr>
</tbody>
</table>
Matrix Components [5.6] Access components of a matrix with array subscripting syntax. For example:

\[
\begin{bmatrix}
  m \\
  m[1] \\
  m[2][0] \\
  m[3][2]
\end{bmatrix}
\]

Examples of operations on matrices and vectors:

\[
m = m \times m; // \text{scalar} \times \text{matrix component-wise}
\]

\[
v = f \times v; // \text{vector} \times \text{vector component-wise}
\]

\[
m = m + m; // \text{matrix component-wise} +/\-
\]

(If m is an array, then m[m] is used to access array components.)

Built-In Functions [8.2]

Component-wise operations. T is float, vec2, vec3, vec4.

\[
\begin{align*}
\text{T pow(T x, T y);} & \quad x^y \\
\text{T exp(T x);} & \quad e^x \\
\text{T log(T x);} & \quad \ln x \\
\text{T log2(T x);} & \quad \log_2 x \\
\text{T sqrt(T x);} & \quad \sqrt x \\
\text{T inversesqrt(T x);} & \quad 1/\sqrt x \\
\text{T abs(T x);} & \quad \text{absolute value} \\
\text{T sign(T x);} & \quad \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \\
\text{T floor(T x);} & \quad \text{nearest integer} \leq x \\
\text{T trunc(T x);} & \quad \text{nearest integer} \leq x \\
\text{T round(T x);} & \quad \text{round} \leq x \\
\text{T roundEven(T x);} & \quad \text{round} \leq x \\
\text{T ceil(T x);} & \quad \text{nearest integer} \geq x \\
\text{T frac(T x);} & \quad x - \text{floor}(x)
\end{align*}
\]

Common Functions [8.3]

Component-wise operations. T is float and vecn, TI is int and ivcn, TU is uint and uvcn, and TB is bool and bvcn. where n is 2, 3, or 4.

\[
\begin{align*}
\text{T max(T x, T y);} & \quad \text{max} \\
\text{T min(T x, T y);} & \quad \text{min} \\
\text{T clamp(T x, T min, T max);} & \quad \text{clamp} \\
\text{T clamp(T x, T min, T max, T maxVal);} & \quad \text{clamp} \\
\text{T clamp(T x, T min, T max, T minVal);} & \quad \text{clamp} \\
\text{T clamp(T x, T minVal, T maxVal);} & \quad \text{clamp} \\
\text{T mix(T x, T y, T a);} & \quad \text{linear blend of} \\
\text{T mix(T x, T y);} & \quad \text{linear blend of} \\
\text{T fract(T x);} & \quad \text{fract} \\
\end{align*}
\]

Exponential Functions [8.2]

Component-wise operation. T is float, vec2, vec3, vec4.

\[
\begin{align*}
\text{T pow(T x, T y);} & \quad x^y \\
\text{T exp(T x);} & \quad e^x \\
\text{T log(T x);} & \quad \ln x \\
\text{T log2(T x);} & \quad \log_2 x \\
\text{T sqrt(T x);} & \quad \sqrt x \\
\text{T abs(T x);} & \quad \text{absolute value} \\
\text{T sign(T x);} & \quad \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \\
\text{T floor(T x);} & \quad \text{nearest integer} \leq x \\
\text{T trunc(T x);} & \quad \text{nearest integer} \leq x \\
\text{T round(T x);} & \quad \text{round} \leq x \\
\text{T roundEven(T x);} & \quad \text{round} \leq x \\
\text{T ceil(T x);} & \quad \text{nearest integer} \geq x \\
\text{T frac(T x);} & \quad x - \text{floor}(x)
\end{align*}
\]

Common Functions [8.3]

Component-wise operation. T is float and vecn, TI is int and ivcn, TU is uint and uvcn, and TB is bool and bvcn. where n is 2, 3, or 4.

\[
\begin{align*}
\text{T max(T x, T y);} & \quad \text{max} \\
\text{T min(T x, T y);} & \quad \text{min} \\
\text{T clamp(T x, T min, T max);} & \quad \text{clamp} \\
\text{T clamp(T x, T min, T max, T maxVal);} & \quad \text{clamp} \\
\text{T clamp(T x, T min, T max, T minVal);} & \quad \text{clamp} \\
\text{T clamp(T x, T minVal, T maxVal);} & \quad \text{clamp} \\
\text{T mix(T x, T y, T a);} & \quad \text{linear blend of} \\
\text{T mix(T x, T y);} & \quad \text{linear blend of} \\
\text{T fract(T x);} & \quad \text{fract} \\
\end{align*}
\]
**Built-In Functions (continued)**

**Common Functions (continued)**

- `smoothStep(T) edge0, T edge1, T x;`
  - clamp and smooth
- `tangent(T x);`
  - true if x is a NaN
- `floatBitsToUInt(T value);`
  - higher integer, preserving float bit level representation
- `uintBitsToFloat(T value);`
  - higher float, preserving integer bit level representation

**Floating-point Pack and Unpack Functions [8.4]**

- `uint packWord16x2(v, vec2 v);`
- `uint packHalf16x2(v, vec2 v);`
- `vec2 unpackWord16x2(v, uint p);`
- `vec2 unpackHalf16x2(v, uint p);`

**Geometric Functions [8.5]**

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

- `float length(T x);`
  - length of vector
- `float distance(T x, T y);`
  - distance between points
- `float dot(T x, T y);`
  - dot product
- `vec3 cross(vec3 x, vec3 y);`
  - cross product
- `T normalize(T x);`
  - normalize vector to length 1
- `T faceforward(N T x, T y);`
  - returns N if dot(N,N) < 0, else -N
- `T reflect(T x, T N);`
  - reflection direction
- `T refract(T x, T N, float r);`
  - refraction vector

**Matrix Functions [8.6]**

Type mat is any matrix type.

- `mat matrixCompMult(mat x, mat y);`
  - multiply x by y component-wise
- `mat2 outerProduct(vec2 v, vec2 r);`
  - linear algebraic column vector
- `mat3 outerProduct(vec3 v, vec2 r);`
  - linear algebraic column vector
- `mat3x2 transpose(mat3x2 m);`
  - transpose of matrix m
- `mat3 transpose(mat3x2 m);`
  - transpose of matrix m
- `mat2x3 transpose(mat2x3 m);`
  - transpose of matrix m
- `mat2x2 transpose(mat2x2 m);`
  - transpose of matrix m
- `mat3 transpose(mat3x3 m);`
  - transpose of matrix m
- `mat4 transpose(mat4x3 m);`
  - transpose of matrix m
- `float determinant(mat2 m);`
  - determinant of matrix m
- `float determinant(mat3 m);`
  - determinant of matrix m
- `float determinant(mat4 m);`
  - determinant of matrix m
- `mat2 inverse(mat2 m);`
  - inverse of matrix m
- `mat3 inverse(mat3 m);`
  - inverse of matrix m
- `mat4 inverse(mat4 m);`
  - inverse of matrix m

**Vector Relational Functions [8.7]**

Compare a x y component-wise. Input and return vector sizes for a particular call must match. Type bvec is bvec2; bvec is bvec3; bvec is bvec4; vec is vec2; vec is vec3; vec is vec4; where n is 2, 3, or 4. T is union of vec and vec2.

- `bvec lessThan(T x, T y);`
  - x < y
- `bvec lessThanEqual(T x, T y);`
  - x <= y
- `bvec greaterThan(T x, T y);`
  - x > y
- `bvec greaterThanEqual(T x, T y);`
  - x >= y
- `bvec equal(T x, T y);`
  - x = y
- `bvec notEqual(T x, T y);`
  - x != y
- `bool any(bvec x);`
  - true if any component of x is true
- `bool all(bvec x);`
  - true if all components of x are true
- `bvec not(bvec x);`
  - logical complement of x

**Texture Lookup Functions [8.8]**

- `vec4 textureSample2D(gSampler2D sampler, int x, int y);`
- `vec4 textureSample3D(gSampler3D sampler, vec3 x, float lod);`
- `vec4 textureSample2DSample(gSampler2DSample sampler, int x, int y);`
- `vec4 textureSample3DSample(gSampler3DSample sampler, vec3 x, float lod);`
- `vec4 textureSample2DSampleShadow(gSampler2DSampleShadow sampler, vec2 x, float lod, i32 xoffset);`
- `vec4 textureSample3DSampleShadow(gSampler3DSampleShadow sampler, vec3 x, float lod, i32 xoffset);`

**Fragment Processing Functions [8.9]**

Approximated using local differentiating.

- `T dfdx(T p);`
  - Derivative in x
- `T dfdy(T p);`
  - Derivative in y
- `T twidht(T p);`
  - abs(df/dx) + abs(df/dy);