**WebGL™** is an immediate-mode 3D rendering API from The Khronos® Group 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 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 khronos.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.io/glesregistry.

- **[n.n.n]** refers to sections in the WebGL 1.0 specification.
- **[n.n.n]** refers to sections in the WebGL 2.0 specification.
- **Content in blue** is newly added with WebGL 2.0.
- **Content in purple** or marked with • has no corresponding OpenGL ES 3.0 function.

## 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 destination alpha operations and compositing with the page.

**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/supersample) and quality. Must obey.

**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 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 by **createBuffer,** bound by **bindBuffer,** destroyed by **deleteBuffer** |
| **WebGLFramebuffer** [5.5] | Created by **createFramebuffer,** bound by **bindFramebuffer,** destroyed by **deleteFramebuffer** |
| **WebGLProgram** [5.6] | Created by **createProgram,** used by **useProgram,** destroyed by **deleteProgram** |
| **WebGLRenderbuffer** [5.7] | Created by **createRenderbuffer,** bound by **bindRenderbuffer,** destroyed by **deleteRenderbuffer** |
| **WebGLShader** [5.8] | Created by **createShader,** attached to program by **attachShader,** destroyed by **deleteShader** |
| **WebGLTexture** [5.9] | Created by **createTexture,** bound by **bindTexture,** destroyed by **deleteTexture** |
| **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:<br>int *size* enum *type* DOMstring *name* |
| **WebGLShaderPrecision-Format** [5.12] | Information returned from calls to **getShaderPrecisionFormat**. The read-only attributes are:<br>int *rangeMin*<br>int *rangeMax*<br>int *precision* |
| **WebGLQuery** [3.2] | Created by **createQuery,** made active by **beginQuery,** concluded by **endQuery,** destroyed by **deleteQuery** |
| **WebGLSampler** [3.3] | Created by **createSampler,** bound by **bindSampler,** destroyed by **deleteSampler** |
| **WebGLSync** [3.4] | Created by **fenceSync,** blocked on by **clientWaitSync,** waited on internal GL by **waitSync,** queried by **getSynciv,** destroyed by **deleteSync** |
| **WebGLTransformFeedback** [3.5] | Created by **createTransformFeedback,** bound by **bindTransformFeedback,** destroyed by **deleteTransformFeedback** |
| **WebGLVertexArrayObject** [3.6] | Created by **createVertexArray,** bound by **bindVertexArray,** destroyed by **deleteVertexArray** |

## WebGL Context Creation [2.1]

To use WebGL, the author must obtain a WebGL rendering context for a given HTMLCanvasElement. This context manages the OpenGL state and renders to the drawing buffer.

[canvas].**getContext**(
"webgl", WebGLContextAttributes? *optionalAttribs*)
Returns a WebGL 1.0 rendering context

[canvas].**getContext**(
"webgl2", WebGLContextAttributes? *optionalAttribs*)
Returns a WebGL 2.0 rendering context

## Per-Fragment Operations [5.14.3]

void **blendColor**(clampf *red*, clampf *green*, clampf *blue*, clampf *alpha*);

void **blendEquation**(enum *mode*);
*mode:* See *modeRGB* for **blendEquationSeparate**

void **blendEquationSeparate**(enum *modeRGB*, enum *modeAlpha*);
*modeRGB*, and *modeAlpha*: FUNC_ADD, FUNC_SUBTRACT, FUNC_REVERSE_SUBTRACT

void **blendFunc**(enum *sfactor*, enum *dfactor*);
*sfactor:* Same as for *dfactor*, plus SRC_ALPHA_SATURATE
*dfactor:* ZERO, ONE, [ONE_MINUS_]SRC_COLOR, [ONE_MINUS_]DST_COLOR, [ONE_MINUS_]SRC_ALPHA, [ONE_MINUS_]DST_ALPHA, [ONE_MINUS_]CONSTANT_COLOR, [ONE_MINUS_]CONSTANT_ALPHA
*sfactor* and *dfactor* may not both reference constant color

void **blendFuncSeparate**(enum *srcRGB*, enum *dstRGB*, enum *srcAlpha*, enum *dstAlpha*);
*srcRGB, srcAlpha*: See *sfactor* for **blendFunc**
*dstRGB, dstAlpha*: See *dfactor* for **blendFunc**

void **depthFunc**(enum *func*);
*func:* NEVER, ALWAYS, LESS, [NOT]EQUAL, {GE, LE}QUAL, GREATER

void **sampleCoverage**(float *value*, bool *invert*);

void **stencilFunc**(enum *func*, int *ref*, uint *mask*);
*func:* NEVER, ALWAYS, LESS, LEQUAL, [NOT]EQUAL, GREATER, GEQUAL

void **stencilFuncSeparate**(enum *face*, enum *func*, int *ref*, uint *mask*);
*face:* FRONT, BACK, FRONT_AND_BACK
*func:* NEVER, ALWAYS, LESS, LEQUAL, [NOT]EQUAL, GREATER, GEQUAL

void **stencilOp**(enum *fail*, enum *zfail*, enum *zpass*);
*fail, zfail,* and *zpass:* KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECR_WRAP

void **stencilOpSeparate**(enum *face*, enum *fail*, enum *zfail*, enum *zpass*);
*face:* FRONT, BACK, FRONT_AND_BACK
*fail, zfail,* and *zpass:* See *fail, zfail,* and *zpass* for **stencilOp**

## ArrayBuffer and Typed Arrays [5.13]

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. Consult the ECMAScript specification for more details on Typed Arrays.

### Buffers

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

### Views

In the following, *ViewType* may be Int8Array, Int16Array, Int32Array, Uint8Array, Uint16Array, Uint32Array, Float32Array.

*ViewType*(ulong *length*);
Creates a view and a new underlying buffer.
*length:* Read-only, number of elements in this view.

*ViewType*(*ViewType other*);
Creates new underlying buffer and copies *other* array.

*ViewType*(type[] *other*);
Creates new underlying buffer and copies *other* array.

*ViewType*(ArrayBuffer *buffer*, [optional] ulong *byteOffset*, [optional] ulong *length*);
Create a new view of given buffer, starting at optional byte offset, extending for optional length elements.
*buffer:* Read-only, buffer backing this view
*byteOffset:* Read-only, byte offset of view start in buffer
*length:* Read-only, number of elements in this view

### Other Properties

*byteLength:* Read-only, length of view in bytes.
const ulong *BYTES_PER_ELEMENT*: element size in bytes.

### Methods

*view*[*i*] = get/set element *i*

**set**(*ViewType other*[, ulong *offset*]);

**set**(type[] *other*[, ulong *offset*]);
Replace elements in this view with those from *other*, starting at optional *offset*.

*ViewType* **subArray**(long *begin*[, long *end*]);
Return a subset of this view, referencing the same underlying buffer.

## Buffer Objects [5.14.5] [3.7.3]

Once bound, buffers may not be rebound with a different target.

void **bindBuffer**(enum *target*, WebGLBuffer? *buffer*);
*target:* ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER, PIXEL_[UN]PACK_BUFFER, COPY_{READ, WRITE}_BUFFER, TRANSFORM_FEEDBACK_BUFFER, UNIFORM_BUFFER

typedef (ArrayBuffer or ArrayBufferView) BufferDataSource

void **bufferData**(enum *target*, long *size*, enum *usage*);
*target:* See *target* for **bindBuffer**
*usage:* STREAM_{DRAW, READ, COPY}, STATIC_{DRAW, READ, COPY}, DYNAMIC_{DRAW, READ, COPY}

void **bufferData**(enum *target*, ArrayBufferView *srcData*, enum *usage*, uint *srcOffset*[, uint *length*=0]);
*target* and *usage:* Same as for **bufferData** above

void **bufferData**(enum *target*, BufferDataSource *data*, enum *usage*);
*target* and *usage:* Same as for **bufferData** above

void **bufferSubData**(enum *target*, long *offset*, BufferDataSource *data*);
*target:* See *target* for **bindBuffer**

void **bufferSubData**(enum *target*, intptr *dstByteOffset*, ArrayBufferView *srcData*, uint *srcOffset*[, uint *length*=0]);
*target:* See *target* for **bindBuffer**

void **copyBufferSubData**(enum *readTarget*, enum *writeTarget*, intptr *readOffset*, intptr *writeOffset*, sizeiptr *size*);

• void **getBufferSubData**(enum *target*, intptr *srcByteOffset*, ArrayBufferView *dstBuffer*[, uint *dstOffset*=0[, uint *length*=0]]);

## Buffer Objects (continued)

Object **createBuffer**();
Corresponding OpenGL ES function is **GenBuffers**

void **deleteBuffer**(WebGLBuffer? *buffer*);

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]

- string[ ] **getSupportedExtensions**();
- object **getExtension**(string *name*);

Available in the **WebGLRenderingContext** interface.

### Get information about the context

- contextStruct **getContextAttributes**();

### 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] [3.7.7]

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

[WebGLHandlesContextLoss] int **getFragDataLocation**(WebGLProgram *program*, DOMString *name*);

void **attachShader**(Object *program*, Object *shader*);

void **bindAttribLocation**(Object *program*, uint *index*, string *name*);

void **compileShader**(Object *shader*);

Object **createProgram**();

Object **createShader**(enum *type*);
*type:* VERTEX_SHADER, FRAGMENT_SHADER

void **deleteProgram**(Object *program*);

void **deleteShader**(Object *shader*);

void **detachShader**(Object *program*, Object *shader*);

Object[ ] **getAttachedShaders**(Object *program*);

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 *pname*);
Corresponding OpenGL ES function is **GetShaderiv**
*pname:* SHADER_TYPE, DELETE_STATUS, COMPILE_STATUS

string **getShaderInfoLog**(Object *shader*);

string **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*);

## Uniforms and Attributes [5.14.10] [3.7.8]

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

void **disableVertexAttribArray**(uint *index*);
*index:* [0, MAX_VERTEX_ATTRIBS - 1]

void **enableVertexAttribArray**(uint *index*);
*index:* [0, MAX_VERTEX_ATTRIBS - 1]

WebGLActiveInfo? **getActiveAttrib**(WebGLProgram *program*, uint *index*);

WebGLActiveInfo? **getActiveUniform**(WebGLProgram *program*, uint *index*);

int **getAttribLocation**(WebGLProgram *program*, string *name*);

any **getUniform**(WebGLProgram? *program*, uint *location*);

WebGLUniformLocation? **getUniformLocation**(Object *program*, string *name*);

any **getVertexAttrib**(uint *index*, enum *pname*);
*pname:* CURRENT_VERTEX_ATTRIB , VERTEX_ATTRIB_ARRAY_{BUFFER_BINDING, ENABLED}, VERTEX_ATTRIB_ARRAY_{NORMALIZED, SIZE, STRIDE, TYPE}, VERTEX_ATTRIB_ARRAY_{INTEGER, DIVISOR}

long **getVertexAttribOffset**(uint *index*, enum *pname*);
Corresponding OpenGL ES function is **GetVertexAttribPointerv**
*pname:* VERTEX_ATTRIB_ARRAY_POINTER

void **uniform[1234]fv**(WebGLUniformLocation? *location*, Float32List *data*[, uint *srcOffset*=0[, uint *srcLength*=0]]);

void **uniform[1234]iv**(WebGLUniformLocation? *location*, Int32List *data*[, uint *srcOffset*=0[, uint *srcLength*=0]]);

**void uniform[1234]uiv**(WebGLUniformLocation? *location*, Uint32List *data*[, uint *srcOffset*=0[, uint *srcLength*=0]]);

**void uniformMatrix[234]fv**(WebGLUniformLocation? *location*, bool *transpose*, Float32List *data*[, uint *srcOffset*=0[, uint *srcLength*=0]]);

**void uniformMatrix[234]x[234]fv**(WebGLUniformLocation? *location*, bool *transpose*, Float32List *data*[, uint *srcOffset*=0[, uint *srcLength*=0]]);

void **vertexAttrib[1234]f**(uint *index*, ...);

void **vertexAttrib[1234]fv**(uint *index*, Array *value*);

void **vertexAttribI4[u]i[v]**(uint *index*, ...);

void **vertexAttribPointer**(uint *index*, int *size*, enum *type*, bool *normalized*, long *stride*, long *offset*);
*type:* BYTE, SHORT, UNSIGNED_{BYTE, SHORT}, FIXED, FLOAT
*index:* [0, MAX_VERTEX_ATTRIBS - 1]
*stride:* [0, 255]
*offset, stride*: must be a multiple of the type size in WebGL

void **vertexAttribIPointer**(uint *index*, int *size*, enum *type*, sizei *stride*, intptr *offset*);

## Special Functions [5.13.3] [3.7.2]

- contextStruct **getContextAttributes**() [5.13.2]

void **disable**(enum *cap*);
*cap:* BLEND, CULL_FACE, DEPTH_TEST, DITHER, POLYGON_OFFSET_FILL, SAMPLE_ALPHA_TO_COVERAGE, SAMPLE_COVERAGE, SCISSOR_TEST, STENCIL_TEST

void **enable**(enum *cap*);
*cap:* See *cap* for **disable**

void **finish**(); [5.13.11]

void **flush**(); [5.13.11]

enum **getError**();
*Returns:* OUT_OF_MEMORY, INVALID_{ENUM, OPERATION, FRAMEBUFFER_OPERATION, VALUE}, NO_ERROR, CONTEXT_LOST_WEBGL

any **getParameter**(enum *pname*);
*pname:* {ALPHA, RED, GREEN, BLUE, SUBPIXEL}_BITS, ACTIVE_TEXTURE, ALIASED_{LINE_WIDTH, POINT_SIZE}_RANGE, ARRAY_BUFFER_BINDING, BLEND_DST_{ALPHA, RGB}, BLEND_EQUATION_{ALPHA, RGB}, BLEND_SRC_{ALPHA, RGB}, BLEND[_COLOR], COLOR_{CLEAR_VALUE, WRITEMASK}, COPY_{READ, WRITE}_BUFFER_BINDING, [NUM_]COMPRESSED_TEXTURE_FORMATS, CULL_FACE[_MODE], CURRENT_PROGRAM, DEPTH_{BITS, CLEAR_VALUE, FUNC}, DEPTH_{RANGE, TEST, WRITEMASK}, DRAW_BUFFERi, DRAW_FRAMEBUFFER_BINDING, ELEMENT_ARRAY_BUFFER_BINDING, DITHER, FRAMEBUFFER_BINDING, FRONT_FACE, FRAGMENT_SHADER_DERIVATIVE_HINT, GENERATE_MIPMAP_HINT, LINE_WIDTH, MAX_3D_TEXTURE_SIZE, MAX_ARRAY_TEXTURE_LAYERS, MAX_CLIENT_WAIT_TIMEOUT_WEBGL, MAX_COLOR_ATTACHMENTS, MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS, MAX_[COMBINED_]TEXTURE_IMAGE_UNITS, MAX_COMBINED_UNIFORM_BLOCKS, MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS, MAX_DRAW_BUFFERS, MAX_ELEMENT_INDEX, MAX_ELEMENTS_{INDICES, VERTICES}, MAX_FRAGMENT_INPUT_COMPONENTS, MAX_FRAGMENT_UNIFORM_{BLOCKS, COMPONENTS}, MAX_PROGRAM_TEXEL_OFFSET, MAX_SAMPLES, MAX_SERVER_WAIT_TIMEOUT, MAX_TEXTURE_LOD_BIAS, MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS, MAX_TRANSFORM_FEEDBACK_SEPARATE_COMPONENTS, MAX_TRANSFORM_FEEDBACK_SEPARATE_ATTRIBS, MAX_UNIFORM_BLOCK_SIZE, MAX_UNIFORM_BUFFER_BINDINGS, MAX_{CUBE_MAP_TEXTURE, RENDERBUFFER, TEXTURE}_SIZE, MAX_VARYING_{COMPONENTS, VECTORS}, MAX_VERTEX_{ATTRIBS, TEXTURE_IMAGE_UNITS}, MAX_VERTEX_UNIFORM_{BLOCKS, COMPONENTS, VECTORS}, MAX_VIEWPORT_DIMS, PACK_ALIGNMENT, MIN_PROGRAM_TEXEL_OFFSET, PACK_ROW_LENGTH, PACK_SKIP_{PIXELS, ROWS}, PIXEL_[UN]PACK_BUFFER_BINDING, POLYGON_OFFSET_{FACTOR, FILL, UNITS}, RASTERIZER_DISCARD, READ_{BUFFER, FRAMEBUFFER_BINDING}, RENDERBUFFER_BINDING, RENDERER, SAMPLE_BUFFERS, SAMPLE_[ALPHA_TO_]COVERAGE, SAMPLE_COVERAGE_{INVERT, VALUE}, SAMPLES, SCISSOR_{BOX, TEST}, SHADING_LANGUAGE_VERSION, STENCIL_{BITS, CLEAR_VALUE, TEST}, STENCIL_[BACK_]{FAIL, FUNC, REF, VALUE_MASK, WRITEMASK}, STENCIL_[BACK_]PASS_DEPTH_{FAIL, PASS}, TEXTURE_BINDING_{2D, CUBE_MAP, 3D, 2D_ARRAY}, TRANSFORM_FEEDBACK_{ACTIVE, BINDING, BUFFER_BINDING}, TRANSFORM_FEEDBACK_PAUSED, UNIFORM_BUFFER_BINDING, UNIFORM_BUFFER_OFFSET_ALIGNMENT, UNPACK_ALIGNMENT, UNPACK_{COLORSPACE_CONVERSION_WEBGL, FLIP_Y_WEBGL, PREMULTIPLY_ALPHA_WEBGL}, UNPACK_IMAGE_HEIGHT, UNPACK_ROW_LENGTH, UNPACK_SKIP_{IMAGES, PIXELS, ROWS}, VENDOR, VERSION, VIEWPORT, VERTEX_ARRAY_BINDING

any **getIndexedParameter**(enum *target*, uint *index*);
*target:* TRANSFORM_FEEDBACK_BUFFER_{BINDING, SIZE, START}, UNIFORM_BUFFER_{BINDING, SIZE, START}

void **hint**(enum *target*, enum *mode*);
*target:* GENERATE_MIPMAP_HINT
*hint:* FASTEST, NICEST, DONT_CARE

bool **isEnabled**(enum *cap*);
*cap:* RASTERIZER_DISCARD Also see *cap* for **disable**

void **pixelStorei**(enum *pname*, int *param*);
*pname:* PACK_ALIGNMENT, PACK_ROW_LENGTH, PACK_SKIP_{PIXELS, ROWS}, UNPACK_ALIGNMENT, UNPACK_COLORSPACE_CONVERSION_WEBGL, UNPACK_{FLIP_Y_WEBGL, PREMULTIPLY_ALPHA_WEBGL}, UNPACK_IMAGE_HEIGHT, UNPACK_ROW_LENGTH, UNPACK_SKIP_{PIXELS, ROWS, IMAGES}

## Rasterization [5.13.3]

void **cullFace**(enum *mode*);
*mode:* BACK, FRONT, FRONT_AND_BACK

void **frontFace**(enum *mode*);
*mode:* CCW, CW

void **lineWidth**(float *width*);

void **polygonOffset**(float *factor*, float *units*);

## 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 Must be <= *zFar*
*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*);

## Detect context lost events [5.13.13]

bool **isContextLost**();

## Writing to the Draw Buffer [5.14.11] [3.7.9]

When rendering is directed to drawing buffer, OpenGL ES rendering calls cause the drawing buffer to be presented to the HTML page compositor at start of next compositing operation.

void **drawArrays**(enum *mode*, int *first*, sizei *count*);
*mode:* POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES
*first:* May not be a negative value.

void **drawElements**(enum *mode*, sizei *count*, enum *type*, intptr *offset*);
*mode:* POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES
*type:* UNSIGNED_BYTE, UNSIGNED_SHORT

void **clear**(bitfield *mask*);

void **vertexAttribDivisor**(uint *index*, uint *divisor*);

void **drawArraysInstanced**(enum *mode*, int *first*, sizei *count*, sizei *instanceCount*);

void **drawElementsInstanced**(enum *mode*, sizei *count*, enum *type*, intptr *offset*, sizei *instanceCount*);

void **drawRangeElements**(enum *mode*, uint *start*, uint *end*, sizei *count*, enum *type*, intptr *offset*);

## 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*);

[WebGLHandlesContextLoss] boolean **isVertexArray**(
WebGLVertexArrayObject? *vertexArray*);

## Read Back Pixels [5.14.12] [3.7.10]

Read pixels in current framebuffer into ArrayBufferView object.

void **readPixels**(int *x*, int *y*, long *width*, long *height*, enum *format*, enum *type*, ArrayBufferView *pixels*);
*format:* RGBA
*type*: UNSIGNED_BYTE

void **readPixels**(int *x*, int *y*, sizei *width*, sizei *height*, enum *format*, enum *type*, ArrayBufferView *dstData*, uint *dstOffset*);

void **readPixels**(int *x*, int *y*, sizei *width*, sizei *height*, enum *format*, enum *type*, intptr *offset*);

## 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_, DRAW_]FRAMEBUFFER

[WebGLHandlesContextLoss] enum **checkFramebufferStatus**(enum *target*);
*target:* [READ_, DRAW_]FRAMEBUFFER
*Returns*: FRAMEBUFFER_{COMPLETE, UNSUPPORTED}, FRAMEBUFFER_INCOMPLETE_{ATTACHMENT, DIMENSIONS, MULTISAMPLE, MISSING_ATTACHMENT}, FRAMEBUFFER_UNDEFINED

Object **createFramebuffer**();
Corresponding OpenGL ES function is **GenFramebuffers**

void **deleteFramebuffer**(Object *buffer*);

void **framebufferRenderbuffer**(enum *target*, enum *attachment*, enum *renderbuffertarget*, WebGLRenderbuffer *renderbuffer*);
*target:* FRAMEBUFFER
*attachment:* COLOR_ATTACHMENT0, COLOR_ATTACHMENT*n* where *n* may be an integer from 1 to 15, {DEPTH, STENCIL, DEPTH_STENCIL}_ATTACHMENT
*renderbuffertarget:* RENDERBUFFER

bool **isFramebuffer**(WebGLFramebuffer *framebuffer*);

void **framebufferTexture2D**(enum *target*, enum *attachment*, enum *textarget*, WebGLTexture *texture*, int *level*);
*target* and *attachment:* Same as for **framebufferRenderbuffer**
*textarget:* TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE{X, Y, Z},

any **getFramebufferAttachmentParameter**(enum *target*, enum *attachment*, enum *pname*);
*target* and *attachment:* Same as for **framebufferRenderbuffer**
*pname:* FRAMEBUFFER_ATTACHMENT_OBJECT_{TYPE, NAME}, FRAMEBUFFER_ATTACHMENT_TEXTURE_LEVEL, FRAMEBUFFER_ATTACHMENT_TEXTURE_CUBE_MAP_FACE, FRAMEBUFFER_ATTACHMENT_{ALPHA, BLUE, GREEN, RED}_SIZE, FRAMEBUFFER_ATTACHMENT_COLOR_ENCODING, FRAMEBUFFER_ATTACHMENT_COMPONENT_TYPE, FRAMEBUFFER_ATTACHMENT_{DEPTH, STENCIL}_SIZE, FRAMEBUFFER_ATTACHMENT_TEXTURE_LAYER

void **blitFramebuffer**(int *srcX0*, int *srcY0*, int *srcX1*, int *srcY1*, int *dstX0*, int *dstY0*, int *dstX1*, int *dstY1*, bitfield *mask*, enum *filter*);

void **framebufferTextureLayer**(enum *target*, enum *attachment*, WebGLTexture? *texture*, int *level*, int *layer*);

void **invalidateFramebuffer**(enum *target*, sequence<enum> *attachments*);

void **invalidateSubFramebuffer** (enum *target*, sequence<enum> *attachments*, int *x*, int *y*, sizei *width*, sizei *height*);

void **readBuffer**(enum *src*);

## 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 *texture*) [5.14.3]
*texture:* [TEXTURE0..TEXTURE*i*] where *i* = MAX_COMBINED_TEXTURE_IMAGE_UNITS - 1

**void bindTexture**(enum *target*, WebGLTexture? *texture*);
*target:* TEXTURE_{2D, 3D, 2D_ARRAY}, TEXTURE_CUBE_MAP

void **copyTexImage2D**(enum *target*, int *level*, enum *internalformat*, int *x*, int *y*, long *width*, long *height*, int *border*);
*target:* TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE_{X,Y,Z} TEXTURE_CUBE_MAP_NEGATIVE_{X,Y,Z}, TEXTURE_3D, TEXTURE_2D_ARRAY
*internalformat:* See Tables 3.12, 3.13, 3.14 in the OpenGL ES 3 specification

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

Object **createTexture**();
Corresponding OpenGL ES function is **GenTextures**

void **deleteTexture**(Object *texture*);

void **generateMipmap**(enum *target*);
*target:* see *target* for **bindTexture**

any **getTexParameter**(enum *target*, enum *pname*);
*target:* TEXTURE_2D, TEXTURE_CUBE_MAP
*pname:* TEXTURE_BASE_LEVEL, TEXTURE_COMPARE_{FUNC, MODE}, TEXTURE_IMMUTABLE_{FORMAT, LEVELS}, TEXTURE_MAX_{LEVEL, LOD}, TEXTURE_MIN_LOD, TEXTURE_{MIN, MAG}_FILTER, TEXTURE_WRAP_{R, S, T}

bool **isTexture**(Object *texture*);

void **texImage2D**(enum *target*, int *level*, enum *internalformat*, long *width*, long *height*, int *border*, enum *format*, enum *type*, ArrayBufferView? *pixels*);
The following values apply to all variations of **texImage2D**.
*target:* See *target* for **copyTexImage2D**
*source:* pixels of type ImageData, image of type HTMLImageElement, canvas of type HTMLCanvasElement, video of type HTMLVideoElement

void **texImage2D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, int *border*, enum *format*, enum *type*, ArrayBufferView *srcData*, uint *srcOffset*);

[throws] void **texImage2D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, int *border*, enum *format*, enum *type*, TexImageSource *source*);

void **texImage2D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, int *border*, enum *format*, enum *type*, intptr *offset*);

void **texParameterf**(enum *target*, enum *pname*, float *param*);
*target:* TEXTURE_2D, TEXTURE_CUBE_MAP
*pname:* TEXTURE_BASE_LEVEL, TEXTURE_COMPARE_{FUNC, MODE}, TEXTURE_MAX_{LEVEL, LOD}, TEXTURE_{MIN, MAG}_FILTER, TEXTURE_MIN_LOD, TEXTURE_WRAP_{R, S, T}

void **texParameteri**(enum *target*, enum *pname*, int *param*);
*target:* TEXTURE_2D, TEXTURE_CUBE_MAP
*pname:* See *pname* for **getTexParameter**

void **texSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, long *width*, long *height*, enum *format*, enum *type*, ArrayBufferView? *pixels*);
Following values apply to all variations of **texSubImage2D**.
*target:* See *target* for **copyTexImage2D**
*format* and *type:* See *format* and *type* for **texImage2D**
*object:* See *object* for **texImage2D**
**texStorage2D** may have lower memory costs than **texImage2D** in some implementations and should be considered a preferred alternative to **texImage2D**.

void **texSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, enum *type*, ArrayBufferView *srcData*, uint *srcOffset*);

void **texSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, enum *type*, TexImageSource *source*);

void **texSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, enum *type*, intptr *offset*);

void **texStorage2D**(enum *target*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*);

void **texStorage3D**(enum *target*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*);
**texStorage3D** may have lower memory costs than **texImage3D** in some implementations and should be considered a preferred alternative to allocate three-dimensional textures.

void **texImage3D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, enum *format*, enum *type*, ArrayBufferView? *srcData*);

void **texImage3D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, enum *format*, enum *type*, ArrayBufferView *srcData*, uint *srcOffset*);

void **texImage3D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, enum *format*, enum *type*, TexImageSource *source*);

void **texImage3D**(enum *target*, int *level*, int *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, enum *format*, enum *type*, intptr *offset*);

void **texSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, enum *type*, ArrayBufferView? *srcData* [, uint *srcOffset*=0]);

void **texSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, enum *type*, TexImageSource *source*);

void **texSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, enum *type*, intptr *offset*);

void **copyTexSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, int *x*, int *y*, sizei *width*, sizei *height*);

void **compressedTexImage2D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, int *border*, ArrayBufferView *srcData*[, uint *srcOffset*=0[, uint *srcLengthOverride*=0]]);

void **compressedTexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, ArrayBufferView *srcData*[, uint *srcOffset*=0[, uint *srcLengthOverride*=0]]);

void **compressedTexImage3D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, ArrayBufferView *srcData*[, uint *srcOffset*=0[, uint *srcLengthOverride*=0]]);

void **compressedTexSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *level*, sizei *level*, sizei *depth*, enum *format*, ArrayBufferView *srcData*[, uint *srcOffset*=0[,uint *srcLengthOverride*=0]]);

void **compressedTexImage2D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, int *border*, sizei *imageSize*, intptr *offset*);

void **compressedTexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*, sizei *imageSize*, intptr *offset*);

void **compressedTexImage3D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, int *border*, sizei *imageSize*, intptr *offset*);

void **compressedTexSubImage3D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, *width*, sizei *height*, sizei *depth*, enum *format*, sizei *imageSize*, intptr *offset*);

## 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**(enum *target*, Object *renderbuffer*);
*target:* RENDERBUFFER

Object **createRenderbuffer**();
Corresponding OpenGL ES function is **GenRenderbuffers**

void **deleteRenderbuffer**(Object *renderbuffer*);

any **getRenderbufferParameter**(enum *target*, enum *pname*);
*target:* RENDERBUFFER
*pname:* RENDERBUFFER_{WIDTH, HEIGHT, INTERNAL_FORMAT}, REDEDRBUFFER_{RED, GREEN, BLUE, ALPHA, DEPTH}_SIZE, RENDERBUFFER_STENCIL_SIZE, RENDERBUFFER_SAMPLES

any **getInternalformatParameter**(enum *target*, enum *internalformat*, enum *pname*);
*pname:* SAMPLES

bool **isRenderbuffer**(Object *renderbuffer*);

void **renderbufferStorage**(enum *target*, enum *internalformat*, sizei *width*, sizei *height*);
*target:* RENDERBUFFER
*internalformat:* Accepts internal formats from OpenGL ES 3.0, as well as DEPTH_STENCIL

void **renderbufferStorageMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*);

## 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.

| Internal Format | Format | Type | Color Renderable | Texture Filterable |
|---|---|---|---|---|
| R8 | RED | UNSIGNED_BYTE | Y | Y |
| R8_SNORM | RED | BYTE | | Y |
| R16F | RED | HALF_FLOAT, FLOAT | **Y** | Y |
| R32F | RED | FLOAT | **Y** | **Y** |
| R8UI | RED_INTEGER | UNSIGNED_BYTE | Y | |
| R8I | RED_INTEGER | BYTE | Y | |
| R16UI | RED_INTEGER | UNSIGNED_SHORT | Y | |
| R16I | RED_INTEGER | SHORT | Y | |
| R32UI | RED_INTEGER | UNSIGNED_INT | Y | |
| R32I | RED_INTEGER | INT | Y | |
| RG8 | RG | UNSIGNED_BYTE | Y | Y |
| RG8_SNORM | RG | BYTE | | Y |
| RG16F | RG | HALF_FLOAT,FLOAT | **Y** | Y |
| RG32F | RG | FLOAT | **Y** | **Y** |
| RG8UI | RG_INTEGER | UNSIGNED_BYTE | Y | |
| RG8I | RG_INTEGER | BYTE | Y | |
| RG16UI | RG_INTEGER | UNSIGNED_SHORT | Y | |
| RG16I | RG_INTEGER | SHORT | Y | |
| RG32UI | RG_INTEGER | UNSIGNED_INT | Y | |
| RG32I | RG_INTEGER | INT | Y | |
| RGB8 | RGB | UNSIGNED_BYTE | Y | Y |
| SRGB8 | RGB | UNSIGNED_BYTE | | Y |
| RGB565 | RGB | UNSIGNED_BYTE, UNSIGNED_SHORT_5_6_5 | Y | Y |
| RGB8_SNORM | RGB | BYTE | | Y |
| R11F_G11F_B10F | RGB | UNSIGNED_INT_10F_11F_11F_REV, HALF_FLOAT, FLOAT | **Y** | Y |
| RGB9_E5 | RGB | UNSIGNED_INT_5_9_9_9_REV, HALF_FLOAT, FLOAT | | Y |
| RGB16F | RGB | HALF_FLOAT, FLOAT | | Y |
| RGB32F | RGB | FLOAT | | **Y** |
| RGB8UI | RGB_INTEGER | UNSIGNED_BYTE | | |
| RGB8I | RGB_INTEGER | BYTE | | |
| RGB16UI | RGB_INTEGER | UNSIGNED_SHORT | | |
| RGB16I | RGB_INTEGER | SHORT | | |
| RGB32UI | RGB_INTEGER | UNSIGNED_INT | | |
| RGB32I | RGB_INTEGER | INT | | |
| RGBA8 | RGBA | UNSIGNED_BYTE | Y | Y |
| SRGB8_ALPHA8 | RGBA | UNSIGNED_BYTE | Y | Y |
| RGBA8_SNORM | RGBA | BYTE | | Y |
| RGB5_A1 | RGBA | UNSIGNED_BYTE, UNSIGNED_SHORT_5_5_5_1, UNSIGNED_INT_2_10_10_10_REV | Y | Y |
| RGBA4 | RGBA | UNSIGNED_BYTE, UNSIGNED_SHORT_4_4_4_4 | Y | Y |
| RGB10_A2 | RGBA | UNSIGNED_INT_2_10_10_10_REV | Y | Y |
| RGBA16F | RGBA | HALF_FLOAT, FLOAT | **Y** | Y |
| RGBA32F | RGBA | FLOAT | **Y** | **Y** |
| RGBA8UI | RGBA_INTEGER | UNSIGNED_BYTE | Y | |
| RGBA8I | RGBA_INTEGER | BYTE | Y | |
| RGB10_A2UI | RGBA_INTEGER | UNSIGNED_INT_2_10_10_10_REV | Y | |
| RGBA16UI | RGBA_INTEGER | UNSIGNED_SHORT | Y | |
| RGBA16I | RGBA_INTEGER | SHORT | Y | |
| RGBA32I | RGBA_INTEGER | INT | Y | |
| RGBA32UI | RGBA_INTEGER | UNSIGNED_INT | Y | |

**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.

**[n.n.n]** and **[Table n.n]** refer to sections and tables in the OpenGL ES Shading Language 3.0 specification at www.khronos.org/registry/gles/

## 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 |
| **isamplerCube** | access integer cube mapped texture |
| **isampler2DArray** | access integer 2D array texture |

### Unsigned Integer Sampler Types (opaque)

| | |
|---|---|
| **usampler2D, usampler3D** | access unsigned integer 2D or 3D texture |
| **usamplerCube** | access unsigned integer cube mapped texture |
| **usampler2DArray** | access unsigned integer 2D array texture |

### Structures and Arrays [4.1.8, 4.1.9]

| | |
|---|---|
| **Structures** | struct *type-name* {<br>  *members*<br>} *struct-name*[]; // optional variable declaration,<br>// optionally an array |
| **Arrays** | float foo[3];<br>Structures, blocks, and structure members can be arrays.<br>Only 1-dimensional arrays supported. |

## 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* *#ifndef* *#else*
*#elif* *#endif* *#error* *#pragma* *#extension* *#line*

**Examples of Preprocessor Directives**

- "#version 300 es" 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 disable; 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 newlines 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 integer 1 if running on an OpenGL-ES Shading Language. |

## Qualifiers

### Storage Qualifiers [4.3]

Variable declarations 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**<br>**centroid in** | Linkage into a shader from a previous stage |
| **out**<br>**centroid out** | 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 procede **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:

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

### Layout Qualifiers [4.3.8]

**layout**(*layout-qualifier*) *block-declaration*
**layout**(*layout-qualifier*) **in/out/uniform**
**layout**(*layout-qualifier*) **in/out/uniform** *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

### Parameter Qualifiers [4.4]

Input values are copied in at function call 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 all color values for any color channel. |

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

## Operators and Expressions

### 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].

| | Operator | Description | Assoc. |
|---|---|---|---|
| 1. | ( ) | parenthetical grouping | N/A |
| 2. | [ ]<br>( )<br>.<br>++ -- | array subscript<br>function call & constructor structure<br>field or method selector, swizzler<br>postfix increment and decrement | L - R |
| 3. | ++ --<br>+ - ~ ! | prefix increment and decrement<br>unary | R - L |
| 4. | * % / | multiplicative | L - R |
| 5. | + - | additive | L - R |
| 6. | << >> | bit-wise shift | L - R |
| 7. | < > <= >= | relational | L - R |
| 8. | == != | equality | L - R |
| 9. | & | bit-wise and | L - R |
| 10. | ^ | bit-wise exclusive or | L - R |
| 11. | \| | bit-wise inclusive or | L - R |
| 12. | && | logical and | L - R |
| 13. | ^^ | logical exclusive or | L - R |
| 14. | \|\| | logical inclusive or | L - R |
| 15. | ? : | selection (Selects an entire operand. Use mix() to select individual components of vectors.) | L - R |
| 16. | =<br>+= -= *= /=<br>%= <<= >>=<br>&= ^= \|= | assignment<br>arithmetic assignments | L - R<br>L - R |
| 17. | , | sequence | L - R |

### Vector Components [5.5]

In addition to array numeric subscript syntax, names of vector components are denoted by a single letter. Components can be swizzled and replicated, e.g.: pos.xx, pos.zy

| | |
|---|---|
| *{x, y, z, w}* | Use when accessing vectors that represent points or normals |
| *{r, g, b, a}* | Use when accessing vectors that represent colors |
| *{s, t, p, q}* | Use when accessing vectors that represent texture coordinates |

## Aggregate Operations and Constructors

### Matrix Constructor Examples [5.4.2]

```
mat2(float)                    // init diagonal
mat2(vec2, vec2);              // column-major order
mat2(float, float,
     float, float);            // column-major order
```

### Structure Constructor Example [5.4.3]

```
struct light {
    float intensity;
    vec3 pos;
};
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

### Matrix Components [5.6]

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

```
mat4 m;          // m represents a matrix
m[1] = vec4(2.0);// sets second column to all 2.0
m[0][0] = 1.0;   // sets upper left element to 1.0
m[2][3] = 2.0;   // sets 4th element of 3rd column to 2.0
```

Examples of operations on matrices and vectors:

```
m = f * m;       // scalar * matrix component-wise
v = f * v;       // scalar * vector component-wise
v = v * v;       // vector * vector component-wise
m = m +/- m;     // matrix component-wise +/-
m = m * m;       // linear algebraic multiply
m = v * m;       // row vector * matrix linear algebraic multiply
m = m * v;       // matrix * column vector linear algebraic multiply
f = dot(v, v);   // vector dot product
v = cross(v, v); // vector cross product
m = matrixCompMult(m, m);   // component-wise multiply
```

(more examples ↗)

### Structure Operations [5.7]

Select structure fields using the period (.) operator. Valid operators are:

| | |
|---|---|
| . | field selector |
| == != | equality |
| = | assignment |

### Array Operations [5.7]

Array elements are accessed using the array subscript operator "[ ]". For example:

```
diffuseColor += lightIntensity[3] * NdotL;
```

The size of an array can be determined using the **.length()** operator. For example:

```
for (i = 0; i < a.length(); i++)
    a[i] = 0.0;
```

## Statements and Structure

### Iteration and Jumps [6]

| | |
|---|---|
| **Entry** | void main() |
| **Iteration** | for (;;) { break, continue }<br>while ( ) { break, continue }<br>do { break, continue } while ( ); |

| | |
|---|---|
| **Jump** | break, continue, return<br>discard // Fragment shader only |
| **Selection** | if ( ) { }<br>if ( ) { } else { }<br>switch ( ) { break, case } |

## Built-In Inputs, Outputs, and Constants [7]

Shader programs use special variables to communicate with fixed-function parts of the pipeline. Output special variables may be read back after writing. Input special variables are read-only. All special variables have global scope.

### Vertex Shader Special Variables [7.1]

**Inputs:**

```
int     gl_VertexID;     // integer index
int     gl_InstanceID;   // instance number
```

**Outputs:**

```
out gl_PerVertex {
    vec4    gl_Position;     // transformed vertex position in clip coordinates
    float   gl_PointSize;    // transformed point size in pixels (point rasterization only)
};
```

### Fragment Shader Special Variables [7.2]

**Inputs:**

```
highp vec4    gl_FragCoord;   // fragment position within frame buffer
bool          gl_FrontFacing; // fragment belongs to a front-facing primitive
mediump vec2  gl_PointCoord;  // 0.0 to 1.0 for each component
```

**Outputs:**

```
highp float   gl_FragDepth;   // depth range
```

### Built-In Constants With Minimum Values [7.3]

| Built-in Constant | Minimum value |
|---|---|
| const mediump int gl_MaxVertexAttribs | 16 |
| const mediump int gl_MaxVertexUniformVectors | 256 |
| const mediump int gl_MaxVertexOutputVectors | 16 |
| const mediump int gl_MaxFragmentInputVectors | 15 |
| const mediump int gl_MaxVertexTextureImageUnits | 16 |
| const mediump int gl_MaxCombinedTextureImageUnits | 32 |
| const mediump int gl_MaxTextureImageUnits | 16 |
| const mediump int gl_MaxFragmentUniformVectors | 224 |
| const mediump int gl_MaxDrawBuffers | 4 |
| const mediump int gl_MinProgramTexelOffset | -8 |
| const mediump int gl_MaxProgramTexelOffset | 7 |

### Built-In Uniform State [7.4]

As an aid to accessing OpenGL ES processing state, the following uniform variables are built into the OpenGL ES Shading Language.

```
struct gl_DepthRangeParameters {
    float near;    // n
    float far;     // f
    float diff;    // f - n
};
uniform gl_DepthRangeParameters  gl_DepthRange;
```

## Built-In Functions

### Angle & Trigonometry Functions [8.1]

Component-wise operation. Parameters specified as *angle* are assumed to be in units of radians. T is float, vec2, vec3, vec4.

| | |
|---|---|
| T **radians** (T *degrees*); | degrees to radians |
| T **degrees** (T *radians*); | radians to degrees |
| T **sin** (T *angle*); | sine |
| T **cos** (T *angle*); | cosine |
| T **tan** (T *angle*); | tangent |
| T **asin** (T *x*); | arc sine |
| T **acos** (T *x*); | arc cosine |
| T **atan** (T *y*, T *x*);<br>T **atan** (T *y_over_x*); | arc tangent |
| T **sinh** (T *x*); | hyperbolic sine |
| T **cosh** (T *x*); | hyperbolic cosine |
| T **tanh** (T *x*); | hyperbolic tangent |
| T **asinh** (T *x*); | arc hyperbolic sine; inverse of sinh |
| T **acosh** (T *x*); | arc hyperbolic cosine; non-negative inverse of cosh |
| T **atanh** (T *x*); | arc hyperbolic tangent; inverse of tanh |

### Exponential Functions [8.2]

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

| | |
|---|---|
| T **pow** (T *x*, T *y*); | $x^y$ |
| T **exp** (T *x*); | $e^x$ |
| T **log** (T *x*); | ln |
| T **exp2** (T *x*); | $2^x$ |
| T **log2** (T *x*); | $\log_2$ |
| T **sqrt** (T *x*); | square root |
| T **inversesqrt** (T *x*); | inverse square root |

### Common Functions [8.3]

Component-wise operation. T is float and vec*n*, TI is int and ivec*n*, TU is uint and uvec*n*, and TB is bool and bvec*n*, where *n* is 2, 3, or 4.

| | |
|---|---|
| T **abs**(T *x*);<br>TI **abs**(TI *x*); | absolute value |
| T **sign**(T *x*);<br>TI **sign**(TI *x*); | returns -1.0, 0.0, or 1.0 |
| T **floor**(T *x*); | nearest integer <= *x* |
| T **trunc** (T *x*); | nearest integer a such that<br>\|*a*\| <= \|*x*\| |
| T **round** (T *x*); | round to nearest integer |
| T **roundEven** (T *x*); | round to nearest integer |
| T **ceil**(T *x*); | nearest integer >= *x* |
| T **fract**(T *x*); | *x* - **floor**(*x*) |

| | |
|---|---|
| T **mod**(T *x*, T *y*);<br>T **mod**(T *x*, float *y*);<br>T **modf**(T *x*, out T *i*); | modulus |
| T **min**(T *x*, T *y*);<br>TI **min**(TI *x*, TI *y*);<br>TU **min**(TU *x*, TU *y*);<br>T **min**(T *x*, float *y*);<br>TI **min**(TI *x*, int *y*);<br>TU **min**(TU *x*, uint *y*); | minimum value |
| T **max**(T *x*, T *y*);<br>TI **max**(TI *x*, TI *y*);<br>TU **max**(TU *x*, TU *y*);<br>T **max**(T *x*, float *y*);<br>TI **max**(TI *x*, int *y*);<br>TU **max**(TU *x*, uint *y*); | maximum value |
| T **clamp**(TI *x*, T *minVal*, T *maxVal*);<br>TI **clamp**(V *x*, TI *minVal*, TI *maxVal*);<br>TU **clamp**(TU *x*, TU *minVal*, TU *maxVal*);<br>T **clamp**(T *x*, float *minVal*, float *maxVal*);<br>TI **clamp**(TI *x*, int *minVal*, int *maxVal*);<br>TU **clamp**(TU *x*, uint *minVal*, uint *maxVal*); | **min**(**max**(*x*, *minVal*), *maxVal*) |
| T **mix**(T *x*, T *y*, T *a*);<br>T **mix**(T *x*, T *y*, float *a*); | linear blend of *x* and *y* |
| T **mix**(T *x*, T *y*, TB *a*); | Selects vector source for each returned component |
| T **step**(T *edge*, T *x*);<br>T **step**(float *edge*, T *x*); | 0.0 if *x* < *edge*, else 1.0 |

(more Common Functions ↗)

## Built-In Functions (continued)

### Common Functions (continued)

| | |
|---|---|
| T **smoothstep**(T *edge0*, T *edge1*, T *x*);<br>T **smoothstep**(float *edge0*, float *edge1*, T *x*); | clamp and smooth |
| TB **isnan**(T *x*); | true if *x* is a NaN |
| TB **isinf**(T *x*); | true if *x* is positive or negative infinity |
| TI **floatBitsToInt**(T *value*);<br>TU **floatBitsToUint**(T *value*); | highp integer, preserving float bit level representation |
| T **intBitsToFloat**(TI *value*);<br>T **uintBitsToFloat**(TU *value*); | highp float, preserving integer bit level representation |

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

| | |
|---|---|
| uint **packSnorm2x16**(vec2 *v*);<br>uint **packUnorm2x16**(vec2 *v*); | convert two floats to fixed point and pack into an integer |
| vec2 **unpackSnorm2x16**(uint *p*);<br>vec2 **unpackUnorm2x16**(uint *p*); | unpack fixed point value pair into floats |
| uint **packHalf2x16**(vec2 *v*); | convert two floats into half-precision floats and pack into an integer |
| vec2 **unpackHalf2x16**(uint *v*); | unpack half value pair into full floats |

### 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 *p0*, T *p1*); | 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**(T *N*, T *I*, T *Nref*); | returns *N* if **dot**(*Nref*, *I*) < 0, else -*N* |
| T **reflect**(T *I*, T *N*); | reflection direction *I* - 2 * **dot**(*N*,*I*) * *N* |
| T **refract**(T *I*, T *N*, float *eta*); | 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 *c*, vec2 *r*);<br>mat3 **outerProduct**(vec3 *c*, vec3 *r*);<br>mat4 **outerProduct**(vec4 *c*, vec4 *r*); | linear algebraic column vector * row vector |
| mat2x3 **outerProduct**(vec3 *c*, vec2 *r*);<br>mat3x2 **outerProduct**(vec2 *c*, vec3 *r*);<br>mat2x4 **outerProduct**(vec4 *c*, vec2 *r*);<br>mat4x2 **outerProduct**(vec2 *c*, vec4 *r*);<br>mat3x4 **outerProduct**(vec4 *c*, vec3 *r*);<br>mat4x3 **outerProduct**(vec3 *c*, vec4 *r*); | linear algebraic column vector * row vector |
| mat2 **transpose**(mat2 *m*);<br>mat3 **transpose**(mat3 *m*);<br>mat4 **transpose**(mat4 *m*);<br>mat2x3 **transpose**(mat3x2 *m*);<br>mat3x2 **transpose**(mat2x3 *m*);<br>mat2x4 **transpose**(mat4x2 *m*);<br>mat4x2 **transpose**(mat2x4 *m*);<br>mat3x4 **transpose**(mat4x3 *m*);<br>mat4x3 **transpose**(mat3x4 *m*); | transpose of matrix *m* |
| float **determinant**(mat2 *m*);<br>float **determinant**(mat3 *m*);<br>float **determinant**(mat4 *m*); | determinant of matrix *m* |
| mat2 **inverse**(mat2 *m*);<br>mat3 **inverse**(mat3 *m*);<br>mat4 **inverse**(mat4 *m*); | inverse of matrix *m* |

### Vector Relational Functions [8.7]

Compare *x* and *y* component-wise. Input and return vector sizes for a particular call must match. Type bvec is bvec*n*; vec is vec*n*; ivec is ivec*n*; uvec is uvec*n*; (where *n* is 2, 3, or 4). T is union of vec and ivec.

| | |
|---|---|
| bvec | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |