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.

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 getActiveInfo
- WebGLShaderPrecisionFormat [5.12] The read-only attributes are: int rangeMin, int rangeMax, int precision

WebGLQuery [3.2] Created by createQuery, made active by beginQuery, destroyed by deleteQuery
- WebGLSampler [3.3] Created by createSampler, bound by bindSampler, destroyed by deleteSampler
- WebGLSync [3.4] Created by fenceSync, blocked on clientWaitSync, waited on internal GL with waitSync, queried by getSync, destroyed by deleteSync
- 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? optionalAttributes; optionalAttributes
  Returns a WebGL 1.0 rendering context
- [canvas].getContext("webgl2").WebGLContextAttributes? optionalAttributes; optionalAttributes
  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 modeRGB);
  * modeRGB: See blendEquation for blendEquationSeparate
  * blendFunc: Same as for fector, plus SRC_ALPHA_SATURATE
  * fector: 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]
  * blendFunc: See blendFunc for blendFuncSeparate
  * depthFunc: See depthFunc for depthFuncSeparate
  * alphaFunc: See alphaFunc for alphaFuncSeparate
  * stencilFunc: See stencilFunc for stencilFuncSeparate
  * frontMask, backMask; frontMask, backMask, stencilMask; frontMask, backMask
  * format: RGBA8, RGB8, RGB5A1, LUMINANCE8, LUMINANCE8_ALPHA8, LUMINANCE16, LUMINANCE16_ALPHA16, LUMINANCE32F, LUMINANCE32F_ALPHA32F, LUMINANCE64F, LUMINANCE64F_ALPHA64F, LUMINANCE128F
  * type: float, uint, int
  * samples: 1, 2, 4, 8, 16

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(along byteLength)
  - 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 along BYTE_PER_ELEMENT: element size in bytes.

Methods
- view[()]; view[setElement 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_UNPACK_BUFFER, COPY_BUFFER, DRAW_BUFFER, TRANSFORM_FEEDBACK_BUFFER, UNKNOWN_BUFFER; type: ArrayBuffer or ArrayBufferView
- void bufferData(enum target, sizeiptr size, data, [optional] int usage);
  * target: TESSARRAY_BUFFER, DYNAMIC_DRAW, READ, COPY, STATIC_DRAW, READ, COPY, DYNAMIC_READ, READ, COPY
  * void bindData(enum target, ArrayBufferView srcData, [optional] uint src Offset, [optional] uint length=0);
  * target and usage: Same as for bufferData above

©2019 The Khronos® Group Inc. - Rev. 0619 www.khronos.org/webgl
### Buffer Objects (continued)

Object createBuffer();

Corresponding OpenGL ES function is `GenBuffers`.

void deleteBuffer(WebGLBuffer? buffer);

generic:

*any* getBufferParameter([enums] enum target, enum pname);

target: `Target for buffer

pname: `BUFFER_SIZE, BUFFER_USAGE`

boolean isBuffer(WebGLBuffer? buffer);

---

### Detect and Enable Extensions [5.14.6]

- `string` [getSupportedExtensions()];

- `object` getExtension(string name);

Available in the WebGLRenderingContext interface.

### Get information about the context

contextStruct getContextAttributes();

### Set and get state

This calls in a 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`).

- `WebGLHandlesContextLost?` getFragmentDataLocation([WebGLProgram? program], DOMString name);

- `void` attachShader(Object program, Object shader);

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

- `void` compileShader([WebGLProgram? program] Object shader);

- `Object` createProgram();

- `Object` createShader([enum] `type`: VERTEX_SHADER, FRAGMENT_SHADER);

- `void` `deleteProgram(Object program);

- `void` `detachShader(Object program, Object shader);

- `Object` `getAttachedShaders(Object program);`

- `any` `getProgramParameter(WebGLProgram? program, enum pname);

- `any` `getProgramParameterInfo(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` shaderProgramParameter(Object shader, enum pname);

- `any` shaderProgramParameterInfo(Object shader, enum pname, String name);

- `any` `getShaderParameter`(`Object program`, `Object shader`, `enum pname`);

- `any` `getShaderSource(Object program, Object shader`);

- `any` `getShaderSourceInfoLog(Object program, Object shader`);

- `any` `getShaderPrecisionParameters(Object program, Object shader, `enum pname`);

- `any` `getShaderUniformParameter(Object program, Object shader, `enum pname`);

- `any` `getShaderUniformParameterInfo(Object program, Object shader, `enum pname`);

- `any` `getShaderUniforms(Object program, Object shader`);

- `any` `getShaderInfoLog(Object program, Object shader`);

- `any` `getShaderExtParameters(Object program, Object shader, `enum pname`);

- `any` `getShaderExtParametersInfo(Object program, Object shader, `enum pname`);

- `any` `getShaderExtInfoLog(Object program, Object shader`);

- `any` `getUniform(Object program, Object shader, `enum pname`);

---

### Rasterization [5.13.3]

- `void` `cullface(enum mode);`

- `mode: BACK, FRONT, FRONT_AND_BACK`

- `void` `frontface(enum mode);`

- `mode: CCW, CW`

- `void` `polyOffset(float factor, float units`);

---

### View and Clip [5.13.3 - 5.13.4]

The viewports specifies the affine transformation of x and y from normalized device coordinate to window coordinates. Drawing buffer size is determined by the `CanvasContext`.

- `void` `depthRange(float zNear, float zFar);`

- `zNear`: Clamped to the range 0 to 1. Must be <= `zFar`.

- `void` `scissor(int x, int y, long width, long height);`

- `void` `viewport(int x, int y, long width, long height);`

---

### 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`: POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES`);

- `first` may not be a negative value.

- `uint` `drawElements([enum] mode, [int] count, [int] index);`

- `polygonOffset([float] factor, [float] units);`

---

### Uniforms and Attributes [5.14.10.3.7.8]

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

- `void` `disableVertexAttribArrays([int] index`);

- `void` `enableVertexAttribArrays([int] index`);

- `WebGLActiveInfo? getActiveAttrib([WebGLProgram? program], `uint` index`);

- `WebGLActiveInfo? getActiveUniform([WebGLProgram? program], `uint` index`);

- `int` `getAttribLocation([WebGLProgram? program], string name`);

- `any` `getUniform([WebGLProgram? program], string name`);

- `any` `getUniformLocation([WebGLProgram? program], string name`);

---

### Detect context lost events [5.13.13]

boolean `isContextLost();`
### Vertex Array Objects [3.7.17]

VAOs encapsulate all state related to the definition of data used by the vertex processor.

```javascript
void bindVertexArray(WebGLVertexArrayObject? vertexArray);
WebGLVertexArrayObject? createVertexArray();
void deleteVertexArray(WebGLVertexArrayObject? vertexArray);
[WebGHLosesContextLoss] boolean isVertexArray(WebGLVertexArrayObject? vertexArray);
```

### Read Back Pixels [3.7.10]

Read pixels in current framebuffer into ArrayBufferView object.

```javascript
void readPixels(int x, int y, long width, long height, enum format, enum type, ArrayBufferView pixels); // RGBA type
void readPixels(int x, int y, size_t width, size_t height, enum format, enum type, ArrayBufferView dstData, int dstOffset); // RGBA type
```

### Texture Objects [5.14.8] [3.7.6]

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

```javascript
void activeTexture(enum texture) [5.14.3] texture: TEXTURE_TEXTURE диапазоне i = MAX_IMPORTED_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 (KX2), TEXTURE_CUBE_MAP_NEGATIVE_X (KX2), 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 xOffs, int yOffs, int x, int y, long width, long height); target: See target for copyTexImage2D
Object createTexture();
Corresponding OpenGL ES function is GenTextures
```

### Framebuffer Objects [5.14.6] [3.7.4]

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

```javascript
void bindFramebuffer(enum target, WebGLFramebuffer? framebuffer); target: READ_FRAMEBUFFER
[WebGHLosesContextLoss] enum checkFramebufferStatus();
void deleteFramebuffer(Object buffer); void framebufferRenderbuffer(enum target, enum attachment, enum renderbufferTarget, WebGLRenderbuffer? renderbuffer); target: FRAMEBUFFER
attachment: COLOR_ATTACHMENT0 - COLOR_ATTACHMENT15, DEPTH_STENCIL, DEPTH_STENCIL_ATTACHMENT
renderbufferTarget: RENDERBUFFER
bool isFramebuffer(WebGLFramebuffer framebuffer);
void framebufferTextureLayer(enum target, enum attachment, WebGLTexture? texture, enum level, int layer);
void invalidateFramebufferObject(enum target, sequence<enum attachments>); void invalidateSubFramebufferObject (enum target, sequence<enum attachments> int x, y, size_t width, size_t height);
void readBufferObject(enum src);
```

### Renderbuffer Objects [5.14.7] [3.7.5]

Renderbuffer objects are used to provide storage for the individual buffers used in a framebuffer object.

```javascript
void bindRenderbufferObject(enum target, Object renderbuffer); target: RENDERBUFFER
Object createRenderbuffer();
Corresponding OpenGL ES function is GenRenderbuffers
```

### Texture [[5.14.8]] [[3.7.6]]

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

```javascript
void activeTexture(enum texture) [5.14.3] texture: TEXTURE_TEXTURE диапазоне i = MAX_IMPORTED_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 (KX2), TEXTURE_CUBE_MAP_NEGATIVE_X (KX2), 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 xOffs, int yOffs, int x, int y, long width, long height); target: See target for copyTexImage2D
Object createTexture();
Corresponding OpenGL ES function is GenTextures
```
Whole Framebuffer Operations [5.14.3]
void clearColor(void)
void clearDepth(void)
void clearStencil(void)
void stencilMaskSeparate(void)
void stencilMask(void)
void stencilMaskSeparate(void)
void stencilMask(void)
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></td>
<td>Y</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>RG8UI</td>
<td>RG_INTEGER</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGBI</td>
<td>RG_INTEGER</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RG16UI</td>
<td>RG_INTEGER</td>
<td>UNSIGNED_SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RG16I</td>
<td>RG_INTEGER</td>
<td>SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RG32UI</td>
<td>RG_INTEGER</td>
<td>UNSIGNED_INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RG32I</td>
<td>RG_INTEGER</td>
<td>INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8</td>
<td>RGB</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SRGB8</td>
<td>RGB</td>
<td>UNSIGNED_BYTE</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>RGB565</td>
<td>RGB</td>
<td>UNSIGNED_BYTE, UNSIGNED_SHORT_5_6_5</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB8_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>RGB16F</td>
<td>RGB</td>
<td>HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32F</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>Y</td>
</tr>
<tr>
<td>RGB8S.SnORM</td>
<td>RGBA</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB5_A1</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE, UNSIGNED_SHORT_5_5_1, UNSIGNED_INT_2_10_10_10_REV</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB4A</td>
<td>RGBA</td>
<td>UNSIGNED_BYTE, UNSIGNED_SHORT_4_4_4_4</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB10_A2</td>
<td>RGBA</td>
<td>UNSIGNED_INT_2_10_10_10_REV</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB16F</td>
<td>RGBA</td>
<td>HALF_FLOAT, FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32F</td>
<td>RGBA</td>
<td>FLOAT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8A.UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB8A.BI</td>
<td>RGBA_INTEGER</td>
<td>BYTE</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB10_A2.UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_INT_2_10_10_10_REV</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RGB16A.UI</td>
<td>RGBA_INTEGER</td>
<td>UNSIGNED_SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB16A.I</td>
<td>RGBA_INTEGER</td>
<td>SHORT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32A.I</td>
<td>RGBA_INTEGER</td>
<td>INT</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RGB32A.UI</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**.

[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, 2x3, 2x4 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 { members }`: optional variable declaration, optionally an array
- **Arrays**
  - `float foo[3]: Structures, blocks, and structure members can be arrays. Only 1-dimensional arrays supported.

### Operators and Expressions
- **Operator** [5.1]
  - Numbered in order of precedence. The relational and equality operators `<> <=` are evaluated to a Boolean. To compare vectors component-wise, use functions such as `lessThan()`, `equal()`, etc. [8.7].

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Assoc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ()</td>
<td>parenthetical grouping</td>
<td>N/A</td>
</tr>
<tr>
<td>2. [ ]</td>
<td>array subscript</td>
<td>L-R</td>
</tr>
<tr>
<td>3. ++</td>
<td>prefix increment and decrement</td>
<td>L-R</td>
</tr>
<tr>
<td>4. * / %</td>
<td>multiplicative</td>
<td>L-R</td>
</tr>
<tr>
<td>5. + -</td>
<td>additive</td>
<td>L-R</td>
</tr>
<tr>
<td>6. &lt;&lt; &gt;&gt;</td>
<td>bit-wise shift</td>
<td>L-R</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>#define</th>
<th>#undef</th>
<th>ifdef</th>
<th>ifndef</th>
<th>else</th>
</tr>
</thead>
</table>

**Examples of Preprocessor Directives**
- "Version 300 es" must appear in the first line of a shader program written in GLSL ES version 3.0. If omitted, the shader will be treated as targeting version 1.0.
- `#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**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LINE</strong></td>
<td>Decimal integer constant that is one more than the number of preceding newlines in the current source string</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>Decimal integer constant that says which source string number is currently being processed.</td>
</tr>
<tr>
<td><strong>VERSION</strong></td>
<td>Decimal integer; e.g., 300</td>
</tr>
<tr>
<td>GL_ES</td>
<td>Defined and set to 1 if running on an OpenGL-ES Shading Language.</td>
</tr>
</tbody>
</table>

### Qualifiers
- **Storage Qualifiers** [4.3]
  - Variable declarations may be preceded by one storage qualifier.

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

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

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

<table>
<thead>
<tr>
<th>Uniform Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform</td>
<td>'restatement of qualifier</td>
</tr>
</tbody>
</table>

- **Layout Qualifiers** [4.3.8]
  - For all shader stages:
    - `layout`: qualifier names. For example:
      `layout(centroid in)`, `layout(centroid out)`, `layout(local), | |

- **Input Layout Qualifiers** [4.3.8.1]
  - For all shader stages:
    - `layout`: `location = integer-constant`

- **Output Layout Qualifiers** [4.3.8.2]
  - For all shader stages:
    - `layout`: `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.

<table>
<thead>
<tr>
<th>Parameter Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>(Default) same as in</td>
</tr>
<tr>
<td>in</td>
<td>For function parameters passed into a function</td>
</tr>
<tr>
<td>out</td>
<td>For function parameters passed back out of a function, but not initialized for use when passed in</td>
</tr>
<tr>
<td>inout</td>
<td>For function parameters passed both into and out of a function</td>
</tr>
</tbody>
</table>

### Precision and Precision Qualifiers [4.5]
- Any floating point, integer, or sampler declaration can have the type preceded by one of these precision qualifiers:

<table>
<thead>
<tr>
<th>Precision Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>highp</td>
<td>Satisfies minimum requirements for the vertex language.</td>
</tr>
<tr>
<td>mediump</td>
<td>Range and precision is between that provided by lowp and highp.</td>
</tr>
<tr>
<td>lowp</td>
<td>Range and precision can be less than mediump, but still represents all color values for any color channel.</td>
</tr>
</tbody>
</table>

### Operators
- **Operators** [5.1]
  - Numbered in order of precedence. The relational and equality operators `<> <=` are evaluated to a Boolean. To compare vectors component-wise, use functions such as `lessThan()`, `equal()`, etc. [8.7].

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Assoc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. ==</td>
<td>equality</td>
<td>L-R</td>
</tr>
<tr>
<td>9. &amp; &amp;</td>
<td>bit-wise and</td>
<td>L-R</td>
</tr>
<tr>
<td>10. ^</td>
<td>bit-wise exclusive or</td>
<td>L-R</td>
</tr>
<tr>
<td>11.</td>
<td></td>
<td>bit-wise inclusive or</td>
</tr>
<tr>
<td>12.</td>
<td>&amp; &amp;</td>
<td>logical and</td>
</tr>
<tr>
<td>13.</td>
<td></td>
<td>logical exclusive or</td>
</tr>
<tr>
<td>14.</td>
<td></td>
<td>logical inclusive or</td>
</tr>
<tr>
<td>15.</td>
<td></td>
<td>selection (Selects an entire operand. Use mask to select individual components of vectors.)</td>
</tr>
<tr>
<td>16.</td>
<td></td>
<td>assignment</td>
</tr>
<tr>
<td>17.</td>
<td></td>
<td>sequence</td>
</tr>
</tbody>
</table>

### 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.: `x, y, z, w` use when accessing vectors that represent points or normals `r, g, b, a` use when accessing vectors that represent colors `x, y, z, w` use when accessing vectors that represent texture coordinates
### Aggregate Operations and Constructors

#### Matrix Constructor Examples [5.4.2]

- `mat2(float)` // init diagonal
- `mat2(vect2, vec2);` // column-major order
- `mat2(float, float, float, float);` // column-major order

#### Structure Constructor Example [5.4.3]

```c
struct light {
    vec3 intensity;
    vec3 direction;
    float attenuation;
};
```

### Statements and Structure

#### Iteration and Jumps [6]

<table>
<thead>
<tr>
<th>Entry</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>void main()</td>
<td></td>
</tr>
</tbody>
</table>

#### Selection

```c
# if (break, continue) while {} else {} switch |
```

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

Shaders 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 g[1].VertexID;` // integer index
- `int g[1].InstanceID;` // instance number

**Outputs:**
- `out gl_PerVertex {
  vec4 g[1].Position;` // transformed vertex position in clip coordinates
- `float g[1].PointSize;` // transformed point size in pixels (point rasterization only)
```
};
```

#### Fragment Shader Special Variables [7.2]

**Inputs:**
- `highp vec4 g[1].FragmentCoord;` // fragment position within frame buffer
- `bool g[1].FrontFacing;` // fragment belongs to a front-facing primitive
- `mediump vec4 g[1].PointCoord;` // for 0.0 to 1.0 for each component

**Outputs:**
- `highp float g[1].FragmentDepth;` // depth range

### 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
- `sin (T angle);` // sine
- `cos (T angle);` // cosine
- `tan (T angle);` // tangent
- `asin (T x);` // arc sine
- `acos (T x);` // arc cosine
- `atan (T y, T x);` // arctangent
- `atan (T y_over_x);` // arctangent
- `sinh (T x);` // hyperbolic sine
- `cosh (T x);` // hyperbolic cosine
- `tanh (T x);` // hyperbolic tangent
- `asinh (T x);` // arc hyperbolic sine; inverse of sinh
- `acosh (T x);` // arc hyperbolic cosine; non-negative inverse of cosh
- `atanh (T x);` // arc hyperbolic tangent; inverse of tanh

#### Exponential Functions [8.2]

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

- `pow (T x, T y);` // x^y
- `exp (T x);` // e^x
- `log (T x);` // ln
- `exp2 (T x);` // 2^x
- `log2 (T x);` // log_2
- `sqrt (T x);` // square root
- `inversesqrt (T x);` // inverse square root

#### Common Functions [8.3]

Component-wise operation. 
T is float and vecn, TI is int and ivecn, 
TU is uint and uvecn, and TB is bool and bvecn

- `abs (T x);` // absolute value
- `sign (T x);` // returns -1.0, 0.0, or 1.0
- `floor (T x);` // nearest integer <= x
- `trunc (T x);` // nearest integer a such that |a| <= |x|
- `round (T x);` // round to nearest integer
- `roundEven (T x);` // round to nearest integer
- `ceil (T x);` // nearest integer >= x
- `frac (T x);` // x - floor(x)

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

<table>
<thead>
<tr>
<th>Built-in Constant</th>
<th>Minimum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>const mediump int gl_MaxVertexAttributes</td>
<td>16</td>
</tr>
<tr>
<td>const mediump int gl_MaxVertexUniformVectors</td>
<td>256</td>
</tr>
<tr>
<td>const mediump int gl_MaxVertexOutputVectors</td>
<td>16</td>
</tr>
<tr>
<td>const mediump int gl_MaxFragmentInputVectors</td>
<td>15</td>
</tr>
<tr>
<td>const mediump int gl_MaxVertexTextureImageUnits</td>
<td>16</td>
</tr>
<tr>
<td>const mediump int gl_MaxCombinedTextureImageUnits</td>
<td>32</td>
</tr>
<tr>
<td>const mediump int gl_MaxTextureImageUnits</td>
<td>16</td>
</tr>
<tr>
<td>const mediump int gl_MaxFragmentUniformVectors</td>
<td>224</td>
</tr>
<tr>
<td>const mediump int gl_MaxDrawBuffers</td>
<td>4</td>
</tr>
<tr>
<td>const mediump int gl_MaxProgramTextOffset</td>
<td>8</td>
</tr>
<tr>
<td>const mediump int gl_MaxProgramTextureOffset</td>
<td>7</td>
</tr>
</tbody>
</table>

#### 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.

```c
struct gl_DepthRangeParameters {
    float near;  // n
    float far;   // f
    float diff;  // f - n
    uniform gl_DepthRangeParameters gl_DepthRange;
```
Built-In Functions (continued) Common Functions (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clamp</td>
<td>clamp x and y component-wise. Input and return vector sizes for a</td>
</tr>
<tr>
<td></td>
<td>particular call must match. Type vecb is bvec; vec is vec; vec4 is</td>
</tr>
<tr>
<td></td>
<td>vec4; vector size is vec4; uvec is uvec; [where n is 2, 3, or 4]; T is</td>
</tr>
<tr>
<td></td>
<td>union of vec and vec4.</td>
</tr>
</tbody>
</table>

Vector Relational Functions [8.7]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lessThan</td>
<td>lessThan(T x, T y);</td>
</tr>
<tr>
<td>lessThanEqual</td>
<td>lessThanEqual(T x, T y);</td>
</tr>
<tr>
<td>greaterThan</td>
<td>greaterThan(T x, T y);</td>
</tr>
<tr>
<td>greaterThanEqual</td>
<td>greaterThanEqual(T x, T y);</td>
</tr>
</tbody>
</table>

Floating-point Pack and Unpack Functions [8.4]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vec2 packNorm2x2</td>
<td>convect two floats to fixed point and pack into an integer.</td>
</tr>
<tr>
<td>vec2 unPackNorm2x2</td>
<td>unpack fixed point value pair into floats.</td>
</tr>
<tr>
<td>vec2 packHalf2x2</td>
<td>convect two floats into half-precision floats and pack into an integer.</td>
</tr>
<tr>
<td>vec2 unPackHalf2x2</td>
<td>unpack half value pair into full floats.</td>
</tr>
</tbody>
</table>

Geometric Functions [8.5]

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

Matrix Functions [8.6]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat matrixMult</td>
<td>multiply x by y component-wise.</td>
</tr>
<tr>
<td>mat2 outerProduct</td>
<td>linear algebraic column vector * row vector.</td>
</tr>
<tr>
<td>mat3 outerProduct</td>
<td>linear algebraic column vector * row vector.</td>
</tr>
<tr>
<td>mat2 transpose</td>
<td>transpose of matrix m</td>
</tr>
<tr>
<td>mat3 transpose</td>
<td>transpose of matrix m</td>
</tr>
<tr>
<td>float determinant</td>
<td>determinant of matrix m</td>
</tr>
<tr>
<td>float inverse</td>
<td>inverse of matrix m</td>
</tr>
</tbody>
</table>

Texture Lookup Functions [8.8]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>textureSize</td>
<td>the function textureSize returns the dimensions of level lod for the</td>
</tr>
<tr>
<td></td>
<td>texture bound to sampler, as described in [2.1.9] of the OpenGL ES 3.0</td>
</tr>
<tr>
<td></td>
<td>specification, under “Texture Size Query”. The initial “W” in a type</td>
</tr>
<tr>
<td></td>
<td>“GRG”, “S”, “RS”, “GRS”, “GRG” or “GRG”.</td>
</tr>
</tbody>
</table>

Fragment Processing Functions [8.9]

Approximated using local differencing.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T dFdx[p]</td>
<td>Derivative in x</td>
</tr>
<tr>
<td>T dFdy[p]</td>
<td>Derivative in y</td>
</tr>
<tr>
<td>T width[p]</td>
<td>abs(dFdx[p]) + abs(dFdy[p])</td>
</tr>
</tbody>
</table>