The WebGL Context and getContext() [2.5]

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

For example:

```html
<canvas id="c"></canvas>
<script type="text/javascript">
  var canvas = document.getElementById("c");
  gl = canvas.getContext("webgl");
  gl.clearColor(1.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);
</script>
</body>
</html>
```

Interfaces

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

WebGLContextAttributes [5.2]

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

Attributes:

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

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

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

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

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

WebGLObject [5.3]

This is the parent interface for all WebGL resource objects.

Resource interface objects:

- **WebGLBuffer** [5.4]
  - Open GL Buffer Object.

- **WebGLProgram** [5.5]
  - OpenGL Program Object.

- **WebGLRenderbuffer** [5.7]
  - OpenGL Renderbuffer Object.

- **WebGlShader** [5.8]
  - OpenGL Shader Object.

- **WebGlTexture** [5.9]
  - OpenGL Texture Object.

- **WebGlUniformLocation** [5.10]
  - Location of a uniform variable in a shader program.

- **WebGlActiveInfo** [5.11]
  - Information returned from calls to getActiveUniform and getActiveUniform. Has the following read-only properties: name, type.

WebGLRenderingContext [5.13.13]

This is the principal interface in WebGL. The functions listed on this reference card are available within this interface.

Attributes:

- **drawingBufferWidth**
  - Type: **GLuint**
  - The actual width of the drawing buffer, which may differ from the width attribute of the HTMLCanvasElement if the implementation is unable to satisfy the requested width or height.

- **drawingBufferHeight**
  - Type: **GLuint**
  - The actual height of the drawing buffer, which may differ from the height attribute of the HTMLCanvasElement if the implementation is unable to satisfy the requested width or height.

ArrayBuffer and Typed Arrays [5.12]

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

Buffers

- **ArrayBuffer**
  - **(ulong)**
  - **length**
  - **byteOffset**
  - **byteLength**
  - **[optional]**
  - **elem**

  Creates a new view of an existing ArrayBuffer. To make the data, create one or more views referencing it.

Views

- **void clear()**
  - **(enum)**
  - **target**
  - **value**
  - **[optional]**
  - **mask**

  Clears the drawing buffer, which may be the entire drawing buffer or a portion of it.

Whole Framebuffer Operations [5.13.3]

- **void bind()**
  - **(GLenum)**
  - **target**
  - **[optional]**
  - **buffer**

  Binds a framebuffer to the current context.

- **void bindFramebuffer()**
  - **(GLenum)**
  - **target**
  - **[optional]**
  - **buffer**
  - **[optional]**
  - **name**

  Binds a framebuffer object to the current context.

Buffer Objects [5.13.5]

- **void deleteBuffer(InteropBuffer buffer)**
  - **(GLenum)**
  - **target**
  - **buffer**

  Deletes a named buffer from the current context.

View and Clip [5.13.3 - 5.13.4]

- **void getFloat()**
  - **(GLenum)**
  - **name**
  - **[optional]**
  - **mode**
  - **[optional]**
  - **target**
  - **[optional]**
  - **usage**

  Retrieves floating-point values from a named buffer.

Rasterization [5.13.3]

- **void getFloat()**
  - **(GLenum)**
  - **name**
  - **[optional]**
  - **mode**

  Retrieves floating-point values from a named buffer.

Detect context lost events [5.13.13]

- **void deleteBuffer(InteropBuffer buffer)**
  - **(GLenum)**
  - **target**
  - **buffer**

  Deletes a named buffer from the current context.
Textures [5.3.8]

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

void activeTexture(enum texture [5.3.3.3] texture: TEXTURE0..TEXTURE_15 where i = MAX_OCCLUSION_attachments - 1)

void bindTexture(enum target, Object texture)

void copyTexImage2D(enum target, int level, enum internalformat, int x, int y, width, height, int border)

void copyTextureSubImage2D(enumer target, int level, int xoffset, int yoffset, int x, int y, width, height, long, long)

Note: The following values apply to all variations of texImage2D.

texture2D: TEXTURE_CUBE_MAP, texture2D: TEXTURE_CUBE_MAP_NEGATIVE_Y, texture2D: TEXTURE_CUBE_MAP_POSITIVE_Z, texture2D: TEXTURE_CUBE_MAP_NEGATIVE_X, texture2D: TEXTURE_CUBE_MAP_POSITIVE_Y

void copyTexImage2D(enumer target, int level, int xoffset, int yoffset, int x, int y, width, height, long, long)

Note: Corresponding OpenGLES function is GenTextures
void deleteTexture(Object texture)

void texImage2D(enumer target, int level, enum internalformat, float[] pixels)

void texImage2D(enumer target, int level, enum internalformat, enum format, enum type, Object object)

void texSubImage2D(enumer target, int level, int xoffset, int yoffset, int x, int y, width, height, long, long)

void texSubImage2D(enumer target, int level, int xoffset, int yoffset, int x, int y, width, height, long, long)

Write to the Draw Buffer [5.3.11.1]

When rendering is directed to drawing buffer, OpenGL ES 2.0 rendering calls cause the drawing buffer to be presented to the HTML page composer at start of next composing operation.

void drawArrays(enumer mode, int first, long count)

void drawElements(enumer mode, long count, enum type, Object object)

Read Back Pixels [5.3.12]

Pixels in the current framebuffer can be read back into an ArrayBufferView object.

void readPixels(x, y, width, height, enum format, enum type, Object pixels)

Framebuffer Objects [5.3.6.3]

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

void bindFramebuffer(enumer target, Object framebuffer)

void checkFramebufferStatus(enumer target)

Framebuffer: Returns: FRAMEBUFFER_COMPLETE, FRAMEBUFFER_INCOMPLETE, FRAMEBUFFER_INCOMPLETE_ATTACHMENT, FRAMEBUFFER_INCOMPLETE_DIMENSIONS, FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT.

Special Functions [5.3.13]

contextStruct getContextAttributes() [5.3.12.3]

void disable(enumer cap) [5.3.13.1]

cap: BLEND, CULL_FACE, DEPTH_TEST, DITHER, POLYGON_OFFSET_FILL, SAMPLE_ALPHA_TO_COVERAGE, SAMPLE_COVERAGE_DEPTH, SAMPLE_COVERAGE_TEST, STENCIL_TEST

void enable(enumer cap) [5.3.13.1]

cap: See cap for disable

void finish() [5.3.13.11]

void flush() [5.3.13.11]

enum getError() [5.3.13.11]

Returns: OUT_OF_MEMORY, INVALID_ENUM, INVALID_OPERATION, FRAMEBUFFER_OPERATION, OPERATION_NO, NO_ERROR, CONTEXT_LOST_WEBGL

any getParameter(enumer pname) [5.3.13.1]


any getProgramParameter(enumer target, enumer program) [5.3.13.1]

program: VERTEX_SHADER, FRAGMENT_SHADER

Hardware Implementations [5.3.14]

Valid Applications [5.3.13.10]

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

void disableVertexAttribArray(enumer index)

index: 0, 1, 2, 3...

void enableVertexAttribArray(enumer index)

index: 0, 1, 2, 3...

void getActiveVertexAttribPointer(Object program, program string)

void getUniformObject(Object program, program string)

void getUniformLocation(Object program, program string)

void getVertexAttribArrayObject(Object program, program string)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)

void getVertexAttribArrayObject(Object program, program string, Object params)
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 OpenGL ES processing pipeline.

[n.n] refers to sections in the OpenGL ES Shading Language 1.0 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
- float: floating scalar
- vec2, vec3, vec4: n-component floating point vector
- bvec2, bvec3, bvec4: signed integer vector
- mat2, mat3, mat4: 2x2, 3x3, 4x4 flat matrix
- sampler2D: access a 2D texture
- samplerCube: access cube mapped texture

Structures and Arrays [4.1.8, 4.1.9]
- Structures: struct type-name {   
  members } struct-name: // optional variable declaration, // optionally an array
- Arrays: vector: float (vec3); * structures and blocks can be arrays  
* only 1-dimensional arrays supported  
* structure members can be arrays

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

Operators
- relational: < >   <=   >=
- arithmetic assignments: + -    !
- logical: & &    ||
- arithmetic: *= /= + + - -
- logical: = =   !=

Examples of operations on matrices and vectors:

Matrix Components [5.6]
Access components of a matrix with an array subscripting syntax. For example:

m + m +/- m; // matrix component-wise addition/subtraction
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

Matrix Constructor Example [5.4]
mat2(float)  // diag 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));

Preprocessor [3.4]
Preprocessor Directives

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

Examples of Preprocessor Directives
- #version: “Version 1.00” in a shader program specifies that the program is written in GLSL ES version 1.00. It is optional. If used, it must occur before anything else in the program other than whitespace or comments.
- #extension: 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>LINE</em></td>
<td>Decimal integer constant that is one more than the number of preceding new-lines in the current source string</td>
</tr>
<tr>
<td><em>VERSION</em></td>
<td>Decimal integer, e.g.: 100</td>
</tr>
</tbody>
</table>

GL_ES Defined and set to integer 1 if running on an OpenGL ES Shading Language.

GL_FRAGMENTPRECISION_HIGH | 1 if high precision is required in the fragment language, else undefined [4.5.4]

Qualifiers [4.3.4]

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

Examples of storage qualifiers:

- none          (Default) local read/write memory, or input parameter
- const         Compile-time constant, or read-only function parameter
- attribute     Linkage between a vertex shader and OpenGL ES for per-vertex data
- uniform       Value does not change across the primitive being processed, uniforms form the linkage between a shader, OpenGL ES, and the application
- varying       Linkage between a vertex shader and fragment shader for interpolated data

Uniform [4.3.4.3]
Use to declare global variables whose values are the same across the entire primitive being processed. All uniform variables are read-only. Use uniform qualifiers with any basic data types, to declare a variable whose type is a structure, or an array of any of these. For example:

vec4 lightPosition;

Varying [4.3.5]
The varying qualifier can be used only with the data types float, vec2, vec3, vec4, mat2, mat3, mat4, or arrays of these. Structures cannot be varying. Varying variables are required to have global scope. Declaration is as follows:

Varying vec3 normal;

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

Parameters may be:

- 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:

Examples of precision qualifiers:

- highp         Satisfies minimum requirements for the vertex language.
- mediump       Satisfies minimum requirements for the fragment language.
- lowp          Range and precision can be less than mediump, but still represents all color values for any color channel.

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:

Operators
- relational: < >   <=   >=
- arithmetic assignments: + -    !
- logical: & &    ||
- arithmetic: *= /= + + - -
- logical: = =   !=

Examples of operations on matrices and vectors:

Aggregate Operations and Constructors [5.4.3]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. []</td>
<td>parenthetical grouping</td>
</tr>
<tr>
<td>2. []</td>
<td>array subscript function call &amp; constructor structure field or method selector, swizzler position increment and decrement</td>
</tr>
<tr>
<td>3. ++</td>
<td>prefix increment and decrement unary</td>
</tr>
<tr>
<td>4. *</td>
<td>multiplicative</td>
</tr>
<tr>
<td>5. +</td>
<td>additive</td>
</tr>
<tr>
<td>6. &lt;=</td>
<td>relational</td>
</tr>
<tr>
<td>7. ==</td>
<td>equality</td>
</tr>
<tr>
<td>8. &amp;&amp;</td>
<td>logical and</td>
</tr>
<tr>
<td>9.</td>
<td></td>
</tr>
<tr>
<td>10. ?:</td>
<td>selection (Selects one entire operand. Use [()[]] to select individual components of vectors)</td>
</tr>
<tr>
<td>11. =</td>
<td>assignment (arithmetically assigns)</td>
</tr>
<tr>
<td>12. ,</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.: pos.xx, pos.yz

- [x, y, z, w] Use when accessing vectors that represent points or normals
- [x, y, z, a] Use when accessing vectors that represent colors
- [t, p, a] Use when accessing vectors that represent texture coordinates

Aggregate Operations and Constructors

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat2(float) // diag diagonal</td>
<td></td>
</tr>
<tr>
<td>mat2(vec2, vec2); // column-major order</td>
<td></td>
</tr>
<tr>
<td>mat2(float, float, float, float); // column-major order</td>
<td></td>
</tr>
</tbody>
</table>

Structure Constructor Example

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

Matrix Components

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

Matrix Constructor Example

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat4(float) // diag diagonal</td>
<td></td>
</tr>
<tr>
<td>mat4(vec2, vec2); // column-major order</td>
<td></td>
</tr>
<tr>
<td>mat4(float, float, float, float); // column-major order</td>
<td></td>
</tr>
</tbody>
</table>

Structure Constructor Example

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

Matrix Components

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

Matrix Constructor Example

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat4(float) // diag diagonal</td>
<td></td>
</tr>
<tr>
<td>mat4(vec2, vec2); // column-major order</td>
<td></td>
</tr>
<tr>
<td>mat4(float, float, float, float); // column-major order</td>
<td></td>
</tr>
</tbody>
</table>

Structure Constructor Example

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

Array Operations [5.4.3]
Array elements are accessed using the array subscript operator [ ] [ ]. For example:

diffuseColor += lightIntensity[3] * NdotL;
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.

- `radians(T)`, `degrees(T)`: degrees to radians
- `sin(T)`, `cos(T)`, `tan(T)`, `asin(T)`, `acos(T)`, `atan(T)`, `atan2(T, y)`
- `cosine` (T) or `cosine(x)`

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

- `exp(T)`, `exp2(T)`, `log2(T)`
- `sqrt(T)`, `square root` (T)
- `inverseSqrt(T)` or `inverse square root` (T)

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

- `abs(T)` or `absolute value` (T)
- `sign(T)` or `sign function` (T)
- `floor(T)` or `round down` (T)
- `ceil(T)` or `ceiling` (T)
- `frac(T)` or `fractional part` (T)
- `mod(T, x)` or `modulus` (T, x)
- `min(T, x)` or `minimum value` (T, x)
- `max(T, x)` or `maximum value` (T, x)
- `clamp(T, T1, T2)` or `clamp` (T, T1, T2)
- `mix(T1, T2, T)`, `minLinear(x, y)` or `linear blend of x and y` (T1, T2, T)
- `step(T, T0)` or `step function` (T, T0)
- `smoothStep(T0, T1, T)` or `smooth step` (T0, T1, T)

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

- `float length(T)` or `length of vector` (T)
- `float distance(T)` or `distance between points` (T, T)
- `float dot(T, T)` or `dot product` (T, T)
- `float cross(vec3, vec3)` or `cross product` (vec3, vec3)
- `T normalize(T)` or `normalize vector to length 1` (T)
- `T reflect(T, N)` or `reflection direction` (T, N)
- `T refract(T, N, eta)` or `refraction vector` (T, N, eta)

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

- `mat I` or `identity matrix` 
- `mat inverse(mat)` or `logical complement` (mat)
- `mat multiply(mat, mat)` or `multiply x by y component-wise` (mat, mat)

**Texture Lookup Functions** [8.7]
Available only in vertex shaders.

- `texture2D(sampler2D, coord)` or `lookup in texture` (sampler2D, coord)
- `textureCube(samplerCube, coord)` or `lookup in cube map` (samplerCube, coord)

**Built-In Constants with Minimum Values** [7.4]

<table>
<thead>
<tr>
<th>Function</th>
<th>Minimum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>const mediump int gl_MaxVertexAttribs</td>
<td>8</td>
</tr>
<tr>
<td>const mediump int gl_MaxVertexUniformBuffers</td>
<td>128</td>
</tr>
<tr>
<td>const mediump int gl_MaxVertexUniforms</td>
<td>8</td>
</tr>
<tr>
<td>const mediump int gl_MaxCombinedVertexUniformBuffers</td>
<td>8</td>
</tr>
<tr>
<td>const mediump int gl_MaxTextureImageUnits</td>
<td>8</td>
</tr>
<tr>
<td>const mediump int gl_MaxFragmentUniformBuffers</td>
<td>16</td>
</tr>
<tr>
<td>const mediump int gl_MaxDrawBuffers</td>
<td>1</td>
</tr>
</tbody>
</table>

**Built-In Uniform State** [7.5]
Specifies depth range in window coordinates. If an implementation does not support high precision in the fragment language, and state is listed as highp, then that state will only be available as mediump in the fragment language.

```cpp
vec4 gl_DepthRangeParameters();
```

**Statements and Structure**
**Iteration and Jumps** [6]

- `call by value-return`
- `for (;;) { break, continue }`
- `if ( ) { } else { }`

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

**Vertex Shader**
```cpp
uniform mat4 mvp_matrix; // model-view-projection matrix
uniform mat3 normal_matrix; // normal matrix
uniform vec3 ec_light_dir; // light direction in eye coords
attribute vec4 a_vertex; // vertex position
attribute vec4 ec_light_dir_a; // ambient lighting
attribute vec2 a_texcoord; // texture coordinates
varying vec3 v_diffuse; // diffuse light component
varying vec2 v_texcoord; // texture coordinates

void main(void)
{
    // put vertex normal into eye coords
    vec3 ec_normal = normalize(normal_matrix * a_vertex);
    // emit diffuse scale factor, texcoord, and position
    v_diffuse = max(dot(ec_light_dir, ec_normal), 0.0) * a_texcoord;
    gl_Position = mvp_matrix * a_vertex;
}
```

**Fragment Shader**
```cpp
precision mediump float;
uniform sampler2D_t t_reflectance;
uniform vec4 i_ambient;

varying vec3 v_diffuse;
varying vec2 v_texcoord;

void main(void)
{
    vec4 color = texture2D(t_reflectance, v_texcoord);
    gl_FragColor = color * (vec4(v_diffuse) + i_ambient);
}
```