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* {
	    members
	} *struct-name*[]; // optional variable declaration,
	                                  // optionally an array

Arrays	float foo[3];
	    Structures, blocks, and structure members can be arrays.
	    Only 1-dimensional arrays supported.

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 **centroid in**	Linkage into a shader from a previous stage
out **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):

				Integer Range	
	FP Range	FP Magnitude Range	FP Precision	Signed	Unsigned
highp	$(-2^{126}, 2^{127})$	$0.0, (2^{-126}, 2^{127})$	Relative 2^{-24}	$[-2^{31}, 2^{31}-1]$	$[0, 2^{32}-1]$
mediump	$(-2^{14}, 2^{14})$	$(2^{-14}, 2^{14})$	Relative 2^{-10}	$[-2^{15}, 2^{15}-1]$	$[0, 2^{16}-1]$
lowp	$(-2, 2)$	$(2^{-8}, 2)$	Absolute 2^{-8}	$[-2^{7}, 2^{7}-1]$	$[0, 2^{8}-1]$

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

    precision **highp** int;

Invariant Qualifiers Examples [4.6]

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

Order of Qualification [4.7]

When multiple qualifications are present, they must follow a strict order. This order is either:

invariant, interpolation, storage, precision

or:

storage, parameter, precision

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.	**[ ]** **( )** **.** **++ --**	array subscript function call & constructor structure field or method selector, swizzler postfix increment and decrement	L - R
3.	**++ --** **+ - ~ !**	prefix increment and decrement 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.	**=**	assignment	L - R
	+= -= *= /= **%= <<= >>=** **&= ^= |=**	arithmetic assignments	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