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.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
int	signed integer
float	floating scalar
vec2, vec3, vec4	n-component floating point vector
bvec2, bvec3, bvec4	Boolean vector
ivec2, ivec3, ivec4	signed integer vector
mat2, mat3, mat4	2x2, 3x3, 4x4 float 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	**float** foo[3]; * structures and blocks can be arrays * only 1-dimensional arrays supported * structure members can be arrays

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.

	Operator	Description	Associativity
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
7.	< > <= >=	relational	L - R
8.	== !=	equality	L - R
12.	&&	logical and	L - R
13.	^^	logical exclusive or	L - R
14.	||	logical inclusive or	L - R
15.	? :	selection (Selects one entire operand. Use mix() to select individual components of vectors.)	L - R
16.	= += -= *= /=	assignment 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

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*	*#version*	*#line*

Examples of Preprocessor Directives

- "#version 100" 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 *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

__LINE__	Decimal integer constant that is one more than the number of preceding new-lines in the current source string
__VERSION__	Decimal integer, e.g.: 100
GL_ES	Defined and set to integer 1 if running on an OpenGL-ES Shading Language.
GL_FRAGMENT_PRECISION_HIGH	1 if highp is supported in the fragment language, else undefined [4.5.4]

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
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]

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:

uniform **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.

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. Optional in the fragment language.
mediump	Satisfies minimum requirements for the fragment language. Its 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.

For example:

lowp float color;
varying mediump vec2 Coord;
lowp ivec2 foo(lowp mat3);
highp mat4 m;

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

	FP Range	FP Magnitude Range	FP Precision	Integer Range
highp	$(-2^{62}, 2^{62})$	$(2^{-62}, 2^{62})$	Relative 2^{-16}	$(-2^{16}, 2^{16})$
mediump	$(-2^{14}, 2^{14})$	$(2^{-14}, 2^{14})$	Relative 2^{-10}	$(-2^{10}, 2^{10})$
lowp	$(-2, 2)$	$(2^{-8}, 2)$	Absolute 2^{-8}	$(-2^{8}, 2^{8})$

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 varying mediump 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 as follows.

invariant, storage, precision
storage, parameter, precision

Aggregate Operations and Constructors

Matrix Constructor Examples [5.4]

```
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 addition/subtraction
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
```

Structure Operations [5.7]

Select structure fields using the period (.) operator. Other operators include:

.	field selector
== !=	equality
=	assignment

Array Operations [4.1.9]

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

diffuseColor += lightIntensity[3] * NdotL;