OpenGL ES 2.0 API Quick Reference Card

OpenGL® ES is a software interface to graphics hardware. The interface consists of a set of procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

- [n.n.n] refers to sections and tables in the OpenGL ES 2.0 specification.
- [n.n.n] refers to sections in the OpenGL ES Shading Language 1.0

Specifications are available at www.opengl.org/registry/gles

Errors [2.5]

enum GetError(void); //Returns one of the following:

and the first property of the first property		
INVALID_ENUM	Enum argument out of range	
INVALID_FRAMEBUFFER_OPERATION	Framebuffer is incomplete	
INVALID_VALUE	Numeric argument out of range	
INVALID_OPERATION	Operation illegal in current state	
OUT_OF_MEMORY	Not enough memory left to execute command	
NO_ERROR	No error encountered	

GL Data Types [2.3]

1

8

8

8

16

16

32

32

32

32

32

32

32

32

ptrbits

ptrbits

Minimum

Bit Width

Description

Signed binary integer

Unsigned binary integer

Unsigned binary integer

Unsigned binary integer

Floating-point value

Characters making up strings

Signed 2's complement binary integer

Signed 2's complement binary integer

Non-negative binary integer size

Enumerated binary integer value

Non-negative binary integer size

Signed 2's complement binary integer

Floating-point value clamped to [0; 1]

Signed 2's complement 16.16 scaled integer

Boolear

GL types are not C types.

GL Type

boolean

byte

ubyte

char

short

ushort

int

uint

fixed

sizei

enum

intptr

sizeiptr

bitfield

float

clampf

OpenGL ES Command Syntax [2.3]

Open GL ES commands are formed from a return type, a name, and optionally a type letter i for 32-bit int, or f for 32-bit float, as shown by the prototype below:

return-type Name{1234}{if}{v} ([args,] T arg1,..., T argN [, args]);

The arguments enclosed in brackets ([args ,] and [, args]) may or may not be present. The argument type T and the number N of arguments may be indicated by the command name suffixes. N is 1, 2, 3, or 4 if present, or else corresponds to the type letters. If "v" is present, an array of N items is passed by a pointer.

For brevity, the OpenGL documentation and this reference may omit the standard prefixes. The actual names are of the forms: glFunctionName(), GL_CONSTANT, GLtype

Buffer Objects [2.9]

Buffer objects hold vertex array data or indices in high-performance server memory.

void GenBuffers(sizei n, uint *buffers); void **DeleteBuffers**(sizei n, const uint *buffers);

Creating and Binding Buffer Objects

void **BindBuffer**(enum *target*, uint *buffer*); *target*: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER

Creating Buffer Object Data Stores

Updating Buffer Object Data Stores void BufferSubData(enum target, intptr offset, sizeiptr size, const void *data);

target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER

Buffer Object Queries [6.1.6, 6.1.3] boolean IsBuffer(uint buffer);

void GetBufferParameteriv(enum target, enum value, T data):

target: ARRAY_BUFFER, ELEMENT_ ARRAY_BUFFER value: BUFFER_SIZE, BUFFER_USAGE

Vertices

Current Vertex State [2.7]

void VertexAttrib{1234}{f}(uint index, T values); void VertexAttrib{1234}{f}v(uint index, T values);

Vertex Arrays [2.8]

Vertex data may be sourced from arrays that are stored in application memory (via a pointer) or faster GPU memory (in a buffer object).

void VertexAttribPointer(uint index, int size, enum type, boolean normalized, sizei stride, const void *pointer); type: BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, FIXED, FLOAT index: [0, MAX_VERTEX_ATTRIBS - 1]

If an ARRAY BUFFER is bound, the attribute will be read from the bound buffer, and pointer is treated as an offset within the buffer.

void EnableVertexAttribArray(uint index);

void DisableVertexAttribArray(uint index); index: [0, MAX_VERTEX_ATTRIBS - 1

void DrawArrays(enum mode, int first, sizei count); void DrawElements(enum mode, sizei count, enum type, void *indices);

mode: POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP,
TRIANGLE FAN, TRIANGLES

type: UNSIGNED BYTE, UNSIGNED SHORT

If an ELEMENT ARRAY BUFFER is bound, the indices will be read from the bound buffer, and indices is treated as an offset within

Rasterization [3]

Point size is taken from the shader builtin gl_PointSize and clamped to the implementation-dependent point size range.

Line Segments [3.4]
void LineWidth(float width);

Polygons [3.5]

void FrontFace(enum dir);

void CullFace(enum mode);
mode: FRONT, BACK, FRONT_AND_BACK

Enable/Disable(CULL_FACE)

void PolygonOffset(float factor, float units); Enable/Disable(POLYGON_OFFSET_FILL)

Pixel Rectangles [3.6, 4.3]

void **PixelStorei**(enum *pname*, int *param*); pname: UNPACK ALIGNMENT, PACK ALIGNMENT

void **BufferData**(enum *target*, sizeiptr *size*, const void **data*, enum *usage*); usage: STATIC_DRAW, STREAM_DRAW, DYNAMIC_DRAW

Viewport and Clipping

Controlling the Viewport [2.12.1 void **DepthRangef**(clampf n, clampf f); void **Viewport**(int x, int y, sizei w, sizei h);

Reading Pixels [4.3.1]
void ReadPixels(int x, int y, sizei width, sizei height,
enum format, enum type, void *data);
format: RGBA type: UNSIGNED_BYTE

Note: ReadPixels() also accepts a queriable implementation-defined *format/type* combination, see [4.3.1].

Texturing [3.7]

Shaders support texturing using at least
MAX_VERTEX_TEXTURE_IMAGE_UNITS images for vertex shaders and at least MAX_TEXTURE_IMAGE_UNITS images for fragment shaders.

void ActiveTexture(enum texture);

texture: [TEXTUREO..TEXTUREi] where i = MAX COMBINED TEXTURE IMAGE UNITS-1

Texture Image Specification [3.7.1]

void Teximage2D(enum target, int level, int internalformat, sizei width, sizei height, int border, enum format, enum type, void *data);

target: TEXTURE 2D, TEXTURE CUBE MAP_POSITIVE_(X,Y,Z)
TEXTURE_CUBE_MAP_NEGATIVE_(X,Y,Z)
internalformat: ALPHA, LUMINANCE, LUMINANCE_ALPHA, RGB,

format: ALPHA, RGB, RGBA, LUMINANCE, LUMINANCE ALPHA type: UNSIGNED_BYTE, UNSIGNED_SHORT_5_6 UNSIGNED_SHORT_4_4_4_4, UNSIGNED_SHORT_5_5_5_1

Conversion from RGBA pixel components to internal texture components:

Base Internal Format	RGBA	Internal Components
ALPHA	Α	A
LUMINANCE	R	L
LUMINANCE _ALPHA	R, A	L, A
RGB	R, G, B	R, G, B
RGBA	R, G, B, A	R, G, B, A

Alt. Texture Image Specification Commands [3.7.2]

Texture images may also be specified using image data taken directly from the framebuffer, and rectangular subregions of existing texture images may be respecified.

void CopyTexImage2D(enum target, int level, enum internalformat, int x, int y, sizei width, sizei height, int border);

target: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z},
TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z} internalformat: See TexImage2D

void **TexSubImage2D**(enum *target*, int *level*, int *xoffset*, int *yoffset*, sizei *width*, sizei *height*, enum *format*,

enum type, void *data); target: TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z} format and type: See TexImage2D

void CopyTexSubImage2D(enum target, int level, int xoffset, int yoffset, int yoffset, int yo, sizei width, sizei height); target: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}

format and type: See TexImage2D

Compressed Texture Images [3.7.3]

void **CompressedTexImage2D**(enum *target*, int *level*, enum *internalformat*, sizei *width*, sizei *height*, int border, sizei imageSize, void *data);

void CompressedTexSubImage2D(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, sizei imageSize, void *data); target and internalformat: See TexImage2D

Texture Parameters [3.7.4]

void TexParameter{if}(enum target, enum pname, T param);

void TexParameter{if}v(enum target, enum pname,

T params); target: TEXTURE_2D, TEXTURE_CUBE_MAP pname: TEXTURE_WRAP_{S, T}, TEXTURE_{MIN, MAG}_FILTER

Manual Mipmap Generation [3.7.11]

void **GenerateMipmap**(enum target); target: TEXTURE 2D, TEXTURE CUBE MAP

Texture Objects [3.7.13]

void BindTexture(enum target, uint texture);

void DeleteTextures(sizei n, uint *textures);

void GenTextures(sizei n, uint *textures);

Enumerated Queries [6.1.3]

void GetTexParameter{if}v(enum target, enum value, target: TEXTURE_2D, TEXTURE_CUBE_MAP

value: TEXTURE_WRAP_{S, T}, TEXTURE_{MIN, MAG}_FILTER

Texture Queries [6.1.4]

boolean IsTexture(uint texture);

Shaders and Programs

Shader Objects [2.10.1]

uint CreateShader(enum type); type: VERTEX_SHADER, FRAGMENT_SHADER

void ShaderSource(uint shader, size count, const char **string, const int *length);

void CompileShader(uint shader);

void ReleaseShaderCompiler(void);

void DeleteShader(uint shader);

Loading Shader Binaries [2.10.2]

void ShaderBinary(sizei count, const uint *shaders, enum binaryformat, const void *binary, sizei length);

Program Objects [2.10.3]

uint CreateProgram(void);

void AttachShader(uint program, uint shader);

void DetachShader(uint program, uint shader);

void LinkProgram(uint program);

void UseProgram(uint program);

void DeleteProgram(uint program);

Shader Variables [2.10.4]

void **GetActiveAttrib**(uint *program*, uint *index*, sizei *bufSize*, sizei **length*, int **size*, enum **type*, char *name);
*type returns: FLOAT, FLOAT VEC{2,3,4}, FLOAT MAT{2,3,4}

int GetAttribLocation(uint program, const char *name);

void BindAttribLocation(uint program, uint index, const char *name);

Uniform Variables

int **GetUniformLocation**(uint *program*, const char *name);

void **GetActiveUniform**(uint *program*, uint *index*, sizei *bufSize*, sizei **length*, int **size*, enum **type*, char *name);

*type: FLOAT, FLOAT_VEC{2,3,4}, INT, INT_VEC{2,3,4}, BOOL, BOOL_VEC{2,3,4}, FLOAT_MAT{2,3,4}, SAMPLER_2D,

void Uniform{1234}{if}(int location, T value);

void Uniform{1234}{if}v(int location, size count, T value);

void UniformMatrix{234}fv(int location, sizei count, boolean transpose, const float *value);

Shader Execution (Validation) [2.10.5] void ValidateProgram(uint program);

Shader Queries Shader Queries [6.1.8]

boolean IsShader(uint shader);

void **GetShaderiv**(uint shader, enum pname, int *params); YPE, DELETE STATUS, COMPILE STATU

void GetAttachedShaders(uint program, sizei maxCount, sizei *count, uint *shaders);

void GetShaderInfoLog(uint shader, sizei bufSize, sizei *length, char *infoLog);

INFO_LOG_LENGTH, SHADER_SOURCE_LENGTH

void GetShaderSource(uint shader, sizei bufSize,

sizei *length, char *source);

void GetShaderPrecisionFormat(enum shadertype, enum precisiontype, int *range, int *precision);

shadertype: VERTEX_SHADER, FRAGMENT_SHADER precision: LOW_FLOAT, MEDIUM_FLOAT, HIGH_FLOAT, LOW_INT, MEDIUM INT, HIGH IN

void GetVertexAttribfv(uint index, enum pname,

float *params); pname: CURRENT_VERTEX_ATTRIB , VERTEX_ATTRIB_ARRAY_x (where x may be BUFFER_BINDING, ENABLED, SIZE, STRIDE, TYPE, NORMALIZED

void GetVertexAttribiv(uint index, enum pname,

int *params);
pname: CURRENT_VERTEX_ATTRIB , VERTEX_ATTRIB_ARRAY_x (where x may be BUFFER_BINDING, ENABLED, SIZE, STRIDE, TYPE, NORMALIZED)

void GetVertexAttribPointerv(uint index, enum pname, void **pointer); pname: VERTEX_ATTRIB_ARRAY_POINTER

void GetUniformfv(uint program, int location, float *params)

void GetUniformiv(uint program, int location, int *params)

Program Queries [6.1.8]

boolean IsProgram(uint program);

void **GetProgramiv**(uint *program*, enum *pname*, int *params); pname: DELETE_STATUS, LINK_STATUS, VALIDATE_STATUS, INFO_LOG_LENGTH, ATTACHED_SHADERS, ACTIVE_ATTRIBUTES, ACTIVE_ATTRIBUTE_MAX_LENGTH,

ACTIVE_UNIFORMS, ACTIVE_UNIFORM_MAX_LENGTH

void GetProgramInfoLog(uint program, sizei bufSize, sizei *length, char *infoLog);

Per-Fragment Operations

Scissor Test [4.1.2] Enable/Disable(SCISSOR_TEST)

void Scissor(int left, int bottom, sizei width, sizei height);

Multisample Fragment Operations [4.1.3]

Enable/Disable(cap)

cap: SAMPLE_ALPHA_TO_COVERAGE, SAMPLE_COVERAGE

void SampleCoverage(clampf value, boolean invert);

Stencil Test [4.1.4]

Enable/Disable(STENCIL TEST)

void **StencilFunc**(enum *func*, int *ref*, uint *mask*);

void StencilFuncSeparate(enum face, enum func, int ref, uint mask):

void **StencilOp**(enum sfail, enum dpfail, enum dppass); void StencilOpSeparate(enum face, enum sfail, enum dpfail,

enum dppass);

face: FRONT, BACK, FRONT_AND_BACK sfail, dpfail, and dppass: KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECR_WRAP

func: NEVER, ALWAYS, LESS, LEQUAL, EQUAL, GREATER, GEQUAL, NOTEQUAL

Depth Buffer Test [4.1.5]

Enable/Disable(DEPTH_TEST)

void **DepthFunc**(enum *Junc*);'

func: NEVER, ALWAYS, LESS, LEQUAL, EQUAL, GREATER, GEQUAL, NOTEQUAL

Blending [4.1.6]

Enable/Disable(BLEND)

(applies to all draw buffers)

void BlendEquation(enum mode);

void BlendEquationSeparate(enum modeRGB,

enum modeAlpha);

mode, modeRGB, and modeAlpha: FUNC_ADD, FUNC_SUBTRACT, FUNC_REVERSE_SUBTRACT

void BlendFuncSeparate(enum srcRGB, enum dstRGB, enum srcAlpha, enum dstAlpha);

void BlendFunc(enum src, enum dst);
dst, dstRGB, and dstAlpha: 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
src, srcRGB, srcAlpha: same for dst, plus SRC_ALPHA_SATURATE

void BlendColor(clampf red, clampf green, clampf blue, clampf alpha);

Dithering [4.1.7

Enable/Disable(DITHER)

Whole Framebuffer Operations

Fine Control of Buffer Updates [4.2.2]

void **ColorMask**(boolean r, boolean q, boolean b, boolean a);

void DepthMask(boolean mask);

void StencilMask(uint mask);

void StencilMaskSeparate(enum face, uint mask);

face: FRONT, BACK, FRONT AND BACK

Clearing the Buffers [4.2.3]

void Clear(bitfield buf); buf: Bitwise OR of COLOR_BUFFER_BIT, DEPTH_BUFFER_BIT,

STENCIL_BUFFER_BIT void ClearColor(clampf r, clampf g, clampf b, clampf a);

void ClearDepthf(clampf d); void ClearStencil(int s);

Special Functions

Flush and Finish [5.1]

Flush guarantees that commands issued so far will eventually complete. Finish blocks until all commands issued so far have

void Flush(void); void Finish(void);

Hint controls certain aspects of GL behavior.

void Hint(enum target, enum hint); target: GENERATE_MIPMAP_HINT hint: FASTEST, NICEST, DONT CARE

State and State Requests

A complete list of symbolic constants for states is shown in the tables in [6.2].

Simple Queries [6.1.1] void GetBooleanv(enum value, boolean *data);

void GetIntegerv(enum value, int *data); void GetFloatv(enum value, float *data); boolean IsEnabled(enum value);

Pointer and String Queries [6.1.5]

ubyte *GetString(enum name); name: VENDOR, RENDERER, VERSION, SHADING_LANGUAGE_VERSION, EXTENSIONS

Framebuffer Objects

Binding & Managing Framebuffer Objects [4.4.1] void BindFramebuffer(enum target, uint framebuffer);

void DeleteFramebuffers(sizei n, uint *framebuffers); void GenFramebuffers(sizei n, uint *framebuffers);

Renderbuffer Objects [4.4.2] void BindRenderbuffer(enum target, uint renderbuffer);

void **DeleteRenderbuffers**(sizei *n*, const uint *renderbuffers); void GenRenderbuffers(sizei n, uint *renderbuffers);

void RenderbufferStorage(enum target,

enum internalformat, sizei width, sizei height);

target: RENDERBUFFER internalformat: DEPTH COMPONENT16, RGBA4, RGB5 A1, RGB565, STENCIL_INDEX8

void FramebufferRenderbuffer(enum target, enum attachment, enum renderbuffertarget, uint renderbuffer):

target: FRAMEBUFFER
attachment: COLOR_ATTACHMENTO, DEPTH_ATTACHMENT,
STENCIL_ATTACHMENT

Attaching Texture Images to a Framebuffer

renderbuffertarget: RENDERBUFFER

void FramebufferTexture2D(enum taraet. enum attachment, enum textarget, uint texture, int level);

textarget: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE{X, Y, Z},
TEXTURE_CUBE_MAP_NEGATIVE{X, Y, Z},
target: FRAMEBUFFER attachment: COLOR_ATTACHMENTO, DEPTH_ATTACHMENT, STENCIL_ATTACHMENT

Framebuffer Completeness [4.4.5]

enum CheckFramebufferStatus(enum target);

target: FRAMEBUFFER

returns: FRAMEBUFFER_COMPLETE or a constant indicating which value violates framebuffer completeness

Framebuffer Object Queries [6.1.3, 6.1.7]

boolean IsFramebuffer(uint framebuffer);

void GetFramebufferAttachmentParameteriv(enum target, enum attachment, enum pname, int *params);

target: Framiediffer attachmento, Depth_attachment, Stencil_attachment attachment: Color_attachment pname: Framebuffer attachment_object_type, Framebuffer attachment_object_name, Framebuffer_attachment_tobject_name, Framebuffer_attachment_texture_level, Framebuffer_attachment_texture_cube_map_face

Renderbuffer Object Queries [6.1.3, 6.1.7] boolean IsRenderbuffer(uint renderbuffer);

void GetRenderbufferParameteriv(enum target,

enum pname, int *params);
target: RENDERBUFFER

target: FRAMEBUFFER

pname: RENDERBUFFER_x (where x may be WIDTH, HEIGHT, RED_SIZE, GREEN_SIZE, BLUE_SIZE, ALPHA_SIZE, DEPTH_SIZE, STENCIL_SIZE, INTERNAL_FORMAT)

The OpenGL® ES Shading Language is two closelyrelated languages which are used to create shaders for the vertex and fragment processors contained in the OpenGL ES processing pipeline.

[n.n.n] and [Table n.n] refer to sections and tables in the OpenGL ES Shading Language 1.0 specification at www.opengl.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.	۸۸	logical exclusive or	L-R
14.	П	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	#praama	#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	
FILE	Decimal integer constant that says which source string number is currently being processed.	
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

none	e (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	

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:

varving 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		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	(-214, 214)	(2 ⁻¹⁴ , 2 ¹⁴)	Relative 2 ⁻¹⁰	(-210, 210)
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 // scalar * vector component-wise

// vector * vector component-wise m = m + /- m;// matrix component-wise addition/subtraction m = m * m; m = v * m; // linear algebraic multiply

// 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;

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]

Outputs:

Variable		Description	Units or coordinate system
highp vec4	gl_Position;	transformed vertex position	clip coordinates
mediump float	gl PointSize;	transformed point size (point rasterization only)	pixels

Fragment Shader Special Variables [7.2]

Fragment shaders may write to gl_FragColor or to one or more elements of gl_FragData[], but not both. The size of the gl_FragData array is given by the built-in constant gl_MaxDrawBuffers.

fragment color for color attachment n

Variable		Description	Units or coordinate system
mediump vec4 gl_FragCoord;		fragment position within frame buffer	window coordinates
bool	gl_FrontFacing;	fragment belongs to a front-facing primitive	Boolean
mediump int gl_PointCoord;		fragment position within a point (point rasterization only)	0.0 to 1.0 for each component
Outputs:			
Variable		Description	Units or coordinate system

Built-In Constants With Minimum Values [7.4]

Built-in Constant	Minimum value
const mediump int gl_MaxVertexAttribs	8
const mediump int gl_MaxVertexUniformVectors	128
const mediump int gl_MaxVaryingVectors	8
const mediump int gl_MaxVertexTextureImageUnits	0
const mediump int gl_MaxCombinedTextureImageUnits	8
const mediump int gl_MaxTextureImageUnits	8
const mediump int gl_MaxFragmentUniformVectors	16
const mediump int gl MaxDrawBuffers	1

Built-In Uniform State [7.5]

Specifies depth range in window coordinates. If an implementation does not support highp precision in the fragment language, and state is listed as highp, then that state will only be available as mediump in the fragment language.

struct gl_DepthRangeParameters {

highp float near;	// n
highp float far;	// f
highp float diff;	// f - r
1.	

uniform gl_DepthRangeParameters gl_DepthRange;

Built-In Functions

mediump vec4 gl_FragColor;

mediump vec4 gl_FragData[n]

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.

fragment color

degrees to radians
radians to degrees
sine
cosine
tangent
arc sine
arc cosine
arc tangent

Exponential Functions [8.2]

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

T pow (T <i>x</i> , T <i>y</i>)	x ^y	
T exp (T <i>x</i>)	ex	
T log(T x)	In	
T exp2 (T <i>x</i>)	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, vec2, vec3, vec4.		
T abs(T x)	absolute value	
T sign(T x)	returns -1.0, 0.0, or 1.0	
T floor(T x)	nearest integer <= x	
T ceil(T x)	nearest integer >= x	
T fract(T x)	x - floor(x)	
T mod (T <i>x</i> , T <i>y</i>) T mod (T <i>x</i> , float <i>y</i>)	modulus	
T min(T x, T y) T min(T x, float y)	minimum value	
T max(T x, T y) T max(T x, float y)	maximum value	
T clamp(T x, T minVal, T maxVal) T clamp(T x, float minVal, float maxVal)	min(max(x, minVal), maxVal)	
T mix (T <i>x</i> , T <i>y</i> , T <i>a</i>) T mix (T <i>x</i> , T <i>y</i> , float <i>a</i>)	linear blend of x and y	
T step(T edge, T x) T step(float edge, T x)	0.0 if <i>x</i> < <i>edge</i> , else 1.0	
T smoothstep(T edge0, T edge1, T x) T smoothstep(float edge0, float edge1, T x)	clip and smooth	

Geometric Functions [8.4]

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

RGBA color

RGBA color

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 [8.5]

Type mat is any matrix type

mat matrixCompMult(mat x, mat y) | multiply x by y component-wise

Vector Relational Functions [8.6]

Compare x and y component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is bvecn; vec is vecn; ivec is ivecn (where n is 2, 3, or 4). T is the union of vec and ivec.

bvec lessThan(T x, T y)	x < y
bvec lessThanEqual(T x, T y)	x <= y
bvec greaterThan(T x, T y)	x > y
bvec greaterThanEqual(T x, T y)	x >= y
bvec equal (T x, T y) bvec equal (bvec x, bvec y)	x == y
bvec notEqual (T x, T y) bvec notEqual (bvec x, bvec y)	x!= y
bool any(bvec x)	true if any component of x is true
bool all(bvec x)	true if all components of x are true
bvec not (bvec x)	logical complement of x

Texture Lookup Functions [8.7]

Available only in vertex shaders

vec4 texture2DLod(sampler2D sampler, vec2 coord, float lod)
vec4 texture2DProjLod(sampler2D sampler, vec3 coord, float lod)
vec4 texture2DProjLod(sampler2D sampler, vec4 coord, float lod)
, ,
vec4 textureCubeLod(samplerCube sampler, vec3 coord, float lod)
Available only in fragment shaders.
vec4 texture2D(sampler2D sampler, vec2 coord, float bias)
ree : tentare == (sample: 25 sample:) reez coora) noac slas)

vec4 texture2DProj(sampler2D sampler, vec3 coord, float bias)

vec4 texture2DProj(sampler2D sampler, vec4 coord, float bias)

vect textureCube(samplerCube sampler vec3 coord float higs)

	vec4 texture cube (sample) cube sumpler, vecs cool a, moat blus
	Available in vertex and fragment shaders.
	vec4 texture2D(sampler2D sampler, vec2 coord)
	vec4 texture2DProj(sampler2D sampler, vec3 coord)
	vec4 texture2DProj(sampler2D sampler, vec4 coord)
	vec4 textureCube(samplerCube sampler, vec3 coord)

Statements and Structure

Iteration and lumns [6]

iteration an	u Juliips [b]
Function Call	call by value-return
Iteration	for (;;) { break, continue } while () { break, continue } do { break, continue } while ();
Selection	if(){} if(){}else{}
Jump	break, continue, return discard // Fragment shader only
Entry	void main()

Sample Program

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

Vertex Shader

```
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 vec3
               a normal;
                                // vertex normal
attribute vec2 a_texcoord;
                                // texture coordinates
varying float v diffuse;
varying vec2 v_texcoord;
void main(void)
```

// put vertex normal into eye coords

```
vec3 ec_normal = normalize(normal_matrix * a_normal);
```

// emit diffuse scale factor, texcoord, and position

```
v diffuse
                  = max(dot(ec_light_dir, ec_normal), 0.0);
v texcoord
                  = a_texcoord;
                  = mvp_matrix * a_vertex;
gl_Position
```

Fragment Shader

```
precision mediump
                     float:
uniform sampler2D t_reflectance;
uniform vec4
                     i_ambient;
varying
         float
                     v_diffuse;
varying
        vec2
                     v_texcoord;
void main (void)
 vec4 color = texture2D(t reflectance, v texcoord);
  gl_FragColor = color * (vec4(v_diffuse) + i_ambient);
```





OpenGL ES is a registered trademark of Silicon Graphics International, used under license by Khronos Group. The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices.

See www.khronos.org to learn more about the Khronos Group. See www.khronos.org/opengles to learn more about OpenGL ES.