

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 3.0 specification.
- **[n.n.n]** refers to sections in the OpenGL ES Shading Language 3.0 specification.

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

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, i64 for int64, f for 32-bit float, or ui for 32-bit uint, as shown by the prototype below:

```
return-type Name{1234}{i i64 f ui}{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. 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: {ELEMENT_ARRAY_BUFFER, PIXEL_UNPACK_BUFFER_COPY, {READ, WRITE}_BUFFER, UNIFORM_BUFFER, TRANSFORM_FEEDBACK_BUFFER}

void **BindBufferRange**(enum target, uint index, uint buffer, intptr offset, sizeiptr size);

target: TRANSFORM_FEEDBACK_BUFFER, UNIFORM_BUFFER

void **BindBufferBase**(enum target, uint index, uint buffer);

target: TRANSFORM_FEEDBACK_BUFFER, UNIFORM_BUFFER

Creating Buffer Object Data Stores

void **BufferData**(enum target, sizeiptr size, const void *data, enum usage);

target: See **BindBuffer**
usage: {STATIC, STREAM, DYNAMIC}_{DRAW, READ, COPY}

void **BufferSubData**(enum target, intptr offset, sizeiptr size, const void *data);

target: See **BindBuffer**

Mapping and Unmapping Buffer Data

void ***MapBufferRange**(enum target, intptr offset, sizeiptr length, bitfield access);

target: See **BindBuffer**
access: Bitwise OR of MAP_{READ, WRITE}_BIT, MAP_INVALIDATE_{RANGE, BUFFER_BIT}, MAP_FLUSH_EXPLICIT_BIT, MAP_UNSYNCHRONIZED_BIT

Vertex Array Objects [2.10, 6.1.10]

void **GenVertexArrays**(sizei n, uint *arrays);

void **DeleteVertexArrays**(sizei n, const uint *arrays);

void **BindVertexArray**(uint array);

boolean **IsVertexArray**(uint array);

Asynchronous Queries [2.13, 6.1.7]

void **GenQueries**(sizei n, uint *ids);

void **BeginQuery**(enum target, uint id);

target: ANY_SAMPLES_PASSED_{CONSERVATIVE}

void **EndQuery**(enum target);

target: ANY_SAMPLES_PASSED_{CONSERVATIVE}

void **DeleteQueries**(sizei n, const uint *ids);

boolean **IsQuery**(uint id);

void **GetQueryiv**(enum target, enum pname, int *params);

void **GetQueryObjectiv**(uint id, enum pname, uint *params);

Transform Feedback [2.14, 6.1.11]

void **GenTransformFeedbacks**(sizei n, uint *ids);

void **DeleteTransformFeedbacks**(sizei n, const uint *ids);

void **BindTransformFeedback**(enum target, uint id);

target: TRANSFORM_FEEDBACK

void **BeginTransformFeedback**(enum primitiveMode);

primitiveMode: TRIANGLES, LINES, POINTS

void **EndTransformFeedback**(void);

Errors [2.5]

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

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

GL Data Types [2.3]

GL types are not C types.

GL Type	Minimum Bit Width	Description
boolean	1	Boolean
byte	8	Signed 2's complement binary integer
ubyte	8	Unsigned binary integer
char	8	Characters making up strings
short	16	Signed 2's complement binary integer
ushort	16	Unsigned binary integer
int	32	Signed 2's complement binary integer
uint	32	Unsigned binary integer
int64	64	Signed 2's complement binary integer
uint64	64	Unsigned binary integer
fixed	32	Signed 2's complement 16.16 scaled integer
sizei	32	Non-negative binary integer size
enum	32	Enumerated binary integer value
intptr	ptrbits	Signed 2's complement binary integer
sizeiptr	ptrbits	Non-negative binary integer size
sync	ptrbits	Sync object handle
bitfield	32	Bit field
half	16	Half-precision float encoded in unsigned scalar
float	32	Floating-point value
clampf	32	Floating-point value clamped to [0, 1]

Viewport and Clipping [2.12.1]

void **DepthRangef**(float n, float f);

void **Viewport**(int x, int y, sizei w, sizei h);

Vertices

Current Vertex State [2.7]

void **VertexAttrib{1234}f**(uint index, float values);

void **VertexAttrib{1234}fv**(uint index, const float *values);

void **VertexAttrib4i**(uint index, T values);

void **VertexAttrib4i**(uint index, const T values);

Vertex Arrays [2.8]

Vertex data may be sourced from arrays stored in client's address space (via a pointer) or in server's address space (in a buffer object).

void **VertexAttribPointer**(uint index, int size, enum type, boolean normalized, sizei stride, const void *pointer);

type: {UNSIGNED}_BYTE, {UNSIGNED}_SHORT, {UNSIGNED}_INT, FIXED, {HALF}_FLOAT, {UNSIGNED}_INT_2_10_10_10_REV
index: [0, MAX_VERTEX_ATTRIBS - 1]

void **VertexAttribIPointer**(uint index, int size, enum type, sizei stride, const void *pointer);

type: {UNSIGNED}_BYTE, {UNSIGNED}_SHORT, {UNSIGNED}_INT
index: [0, MAX_VERTEX_ATTRIBS - 1]

void **EnableVertexAttribArray**(uint index);

void **DisableVertexAttribArray**(uint index);

void **VertexAttribDivisor**(uint index, uint divisor);

void **Enable**(enum target);

void **Disable**(enum target);

target: PRIMITIVE_RESTART_FIXED_INDEX

Drawing [2.8.3]

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

void **DrawArraysInstanced**(enum mode, int first, sizei count, sizei primcount);

void **DrawElements**(enum mode, sizei count, enum type, const void *indices);

type: UNSIGNED_BYTE, UNSIGNED_SHORT, UNSIGNED_INT

void **DrawElementsInstanced**(enum mode, sizei count, enum type, const void *indices, sizei primcount);

type: UNSIGNED_BYTE, UNSIGNED_SHORT, UNSIGNED_INT

void **DrawRangeElements**(enum mode, int start, int end, sizei count, enum type, const void *indices);

mode: POINTS, TRIANGLES, LINES, LINE_{STRIP, LOOP}, TRIANGLE_STRIP, TRIANGLE_FAN
type: UNSIGNED_BYTE, UNSIGNED_SHORT, UNSIGNED_INT

void **FlushMappedBufferRange**(enum target, intptr offset, sizeiptr length);

target: See **BindBuffer**

boolean **UnmapBuffer**(enum target);

target: See **BindBuffer**

Copying Between Buffers

void **CopyBufferSubData**(enum readtarget, enum writetarget, intptr readoffset, intptr writeoffset, sizeiptr size);

readtarget, writetarget: See target for **BindBuffer**

Buffer Object Queries [6.1.9]

boolean **IsBuffer**(uint buffer);

void **GetBufferParameteriv**(enum target, enum pname, int *data);

target: See **BindBuffer**
pname: BUFFER_{SIZE, USAGE, ACCESS_FLAGS, MAPPED}, BUFFER_MAP_{POINTER, OFFSET, LENGTH}

void **GetBufferParameteri64v**(enum target, enum pname, int64 *data);

target, pname: See **GetBufferParameteriv**

void **GetBufferPointerv**(enum target, enum pname, void **params);

target: See **BindBuffer**
pname: BUFFER_MAP_POINTER

Reading and Copying Pixels [4.3.1-2]

void **ReadPixels**(int x, int y, sizei width, sizei height, enum format, enum type, void *data);

format: RGBA, RGBA_INTEGER
type: INT, UNSIGNED_INT_2_10_10_10_REV, UNSIGNED_BYTE, INT
Note: **ReadPixels()** also accepts a queryable implementation-chosen format/type combination [4.3.1].

void **ReadBuffer**(enum src);

src: BACK, NONE, or COLOR_ATTACHMENTi where i may range from zero to the value of MAX_COLOR_ATTACHMENTS - 1

void **BlitFramebuffer**(int srcX0, int srcY0, int srcX1, int srcY1, int dstX0, int dstY0, int dstX1, int dstY1, bitfield mask, enum filter);

mask: Bitwise OR of {COLOR, DEPTH, STENCIL}_BUFFER_BIT
filter: LINEAR or NEAREST

Rasterization [3]

Points [3.4]

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

Line Segments [3.5]

void **LineWidth**(float width);

Polygons [3.6]

void **FrontFace**(enum dir);

dir: CCW, CW

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);

void **PauseTransformFeedback**(void);

void **ResumeTransformFeedback**(void);

boolean **IsTransformFeedback**(uint id);

Shaders and Programs

Shader Objects [2.11.1]

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

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

void CompileShader(uint shader);

void ReleaseShaderCompiler(void);

void DeleteShader(uint shader);
```

Loading Shader Binaries [2.11.2]

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

Program Objects [2.11.3-4]

```
uint CreateProgram(void);

void AttachShader(uint program, uint shader);

void DetachShader(uint program, uint shader);

void LinkProgram(uint program);

void UseProgram(uint program);

void ProgramParameteri(uint program, enum pname,
    int value);
    pname: PROGRAM_BINARY_RETRIEVABLE_HINT

void DeleteProgram(uint program);

void GetProgramBinary(uint program, sizei bufSize,
    sizei *length, enum *binaryFormat, void *binary);

void ProgramBinary(uint program, enum binaryFormat,
    const void *binary, sizei length);
```

Vertex Attributes [2.11.5]

```
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},
    FLOAT_MAT{2x3, 2x4, 3x2, 3x4, 4x2, 4x3}, {UNSIGNED}_JINT,
    {UNSIGNED}_JINT_VEC{2,3,4}

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

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

Uniform Variables [2.11.6]

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

uint GetUniformBlockIndex(uint program,
    const char *uniformBlockName);
```

```
void GetActiveUniformBlockName(uint program,
    uint uniformBlockIndex, sizei bufSize, sizei *length,
    char *uniformBlockName);

void GetActiveUniformBlockiv(uint program,
    uint uniformBlockIndex, enum pname, int *params);
    pname: UNIFORM_BLOCK_BINDING, DATA_SIZE, NAME_LENGTH,
    UNIFORM_BLOCK_ACTIVE_{UNIFORMS, UNIFORM_INDICES},
    UNIFORM_BLOCK_REFERENCED_BY_{VERTEX, FRAGMENT}_SHADER

void GetUniformIndices(uint program, sizei uniformCount,
    const char * const *uniformNames, uint *uniformIndices);

void GetActiveUniform(uint program, uint uniformIndex,
    sizei bufSize, sizei *length, int *size, enum *type,
    char *name);
    *type returns: FLOAT, BOOL, {FLOAT, BOOL}_VEC{2, 3, 4},
    {UNSIGNED}_JINT, {UNSIGNED}_JINT_VEC{2, 3, 4}, FLOAT_MAT{2, 3, 4},
    FLOAT_MAT{2x3, 2x4, 3x2, 3x4, 4x2, 4x3}, SAMPLER_{2D, 3D},
    SAMPLER_{CUBE_SHADOW}, SAMPLER_2D_{ARRAY_SHADOW},
    {UNSIGNED}_JINT_SAMPLER_{2D, 3D, CUBE},
    {UNSIGNED}_JINT_SAMPLER_2D_ARRAY

void GetActiveUniformsiv(uint program, sizei uniformCount,
    const uint *uniformIndices, enum pname, int *params);
    pname: UNIFORM_TYPE, UNIFORM_SIZE, UNIFORM_NAME_LENGTH,
    UNIFORM_BLOCK_INDEX, UNIFORM_{OFFSET, ARRAY_STRIDE},
    UNIFORM_MATRIX_STRIDE, UNIFORM_IS_ROW_MAJOR

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

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

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

void Uniform{1234}uiv(int location, sizei count, const T value);

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

void UniformMatrix{2x3, 3x2, 2x4, 4x2, 3x4, 4x3}fv(
    int location, sizei count, boolean transpose,
    const float *value);

void UniformBlockBinding(uint program,
    uint uniformBlockIndex, uint uniformBlockBinding);
```

Output Variables [2.11.8]

```
void TransformFeedbackVaryings(uint program, sizei count,
    const char * const *varyings, enum bufferMode);
    bufferMode: INTERLEAVED_ATTRIBUTES, SEPARATE_ATTRIBUTES

void GetTransformFeedbackVarying(uint program,
    uint index, sizei bufSize, sizei *length, sizei *size,
    enum *type, char *name);
    *type returns any of the scalar, vector, or matrix attribute types
    returned by GetActiveAttrib().
```

```
type: {UNSIGNED}_BYTE, {UNSIGNED}_SHORT, {UNSIGNED}_INT,
    {HALF}_FLOAT, UNSIGNED_SHORT_4_4_4_4,
    UNSIGNED_SHORT_5_5_5_1, UNSIGNED_SHORT_5_6_5,
    UNSIGNED_INT_2_10_10_10_REV, UNSIGNED_INT_24_8,
    UNSIGNED_INT_10F_11F_11F_REV, UNSIGNED_INT_5_9_9_9_REV,
    FLOAT_32_UNSIGNED_INT_24_8_REV

internalformat: R8, R8I, R8UI, R8_SNORM, R16I, R16UI, R16F, R32I,
    R32UI, R32F, RG8, RG8I, RG8UI, RG8_SNORM, RG16I, RG16UI, RG16F,
    RG32I, RG32UI, RG32F, RGB, RGB5_A1, RGB565, RGB8, RGB8I,
    RGB8UI, RGB8_SNORM, RGB9_E5, RGB10_A2, RGB10_A2UI, RGB16I,
    RGB16UI, RGB16F, RGB32I, RGB32UI, RGB32F, SRGB8, RGBA, RGBA4,
    RGBA8, RGBA8I, RGBA8UI, RGBA8_SNORM, RGBA16I, RGBA16UI,
    RGBA16F, RGBA32I, RGBA32UI, RGBA32F, SRGB8_ALPHA8,
    R11F_G11F_B10F, DEPTH_COMPONENT16, DEPTH_COMPONENT24,
    DEPTH_COMPONENT32F, DEPTH24_STENCIL8, DEPTH32F_STENCIL8,
    LUMINANCE_ALPHA, LUMINANCE, ALPHA
```

```
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, NEGATIVE}_{X, Y, Z}
```

```
internalformat: See TexImage3D
format, type: See TexImage3D
```

```
void TexStorage2D(enum target, sizei levels,
    enum internalformat, sizei width, sizei height);
```

```
target: TEXTURE_CUBE_MAP, TEXTURE_2D
internalformat: See TexImage3D except for unsized base internal
formats in Table 3.3
```

```
void TexStorage3D(enum target, sizei levels,
    enum internalformat, sizei width, sizei height,
    sizei depth);
```

```
target: TEXTURE_3D, TEXTURE_2D_ARRAY
internalformat: See TexImage3D except for unsized base internal
formats in Table 3.3
```

Alt. Texture Image Specification Commands [3.8.5]

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

Shader Execution [2.11.9, 3.9.2]

```
void ValidateProgram(uint program);

int GetFragDataLocation(uint program,
    const char *name);
```

Shader Queries

Shader Queries [6.1.12]

```
boolean IsShader(uint shader);

void GetShaderiv(uint shader, enum pname,
    int *params);
    pname: SHADER_TYPE, {VERTEX, FRAGMENT_SHADER},
    {DELETE, COMPILE}_STATUS, INFO_LOG_LENGTH,
    SHADER_SOURCE_LENGTH

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

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

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_INT

void GetVertexAttribfv(uint index, enum pname,
    float *params);
    pname: CURRENT_VERTEX_ATTRIB, VERTEX_ATTRIB_ARRAY_x
    (where x may be BUFFER_BINDING, DIVISOR, ENABLED,
    INTEGER, SIZE, STRIDE, TYPE, NORMALIZED)

void GetVertexAttribiv(uint index, enum pname,
    int *params);
    pname: See GetVertexAttribfv\(\)

void GetVertexAttribIiv(uint index, enum pname,
    int *params);
    pname: See GetVertexAttribfv\(\)

void GetVertexAttribIuiv(uint index, enum pname,
    int *params);
    pname: See GetVertexAttribfv\(\)

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);

void GetUniformuiv(uint program, int location,
    uint *params);
```

Program Queries [6.1.12]

```
boolean IsProgram(uint program);

void GetProgramiv(uint program, enum pname,
    int *params);
    pname: {DELETE, LINK, VALIDATE}_STATUS,
    INFO_LOG_LENGTH, TRANSFORM_FEEDBACK_VARYINGS,
    TRANSFORM_FEEDBACK_BUFFER_MODE, VARYINGS,
    TRANSFORM_FEEDBACK_VARYING_MAX_LENGTH,
    ATTACHED_SHADERS, ACTIVE_{ATTRIBUTES, UNIFORMS},
    ACTIVE_{ATTRIBUTE, UNIFORM}_MAX_LENGTH,
    ACTIVE_UNIFORM_BLOCK_MAX_NAME_LENGTH,
    PROGRAM_BINARY_RETRIEVABLE_HINT,
    ACTIVE_UNIFORM_BLOCKS

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

```
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 TexImage3D, except for DEPTH* values
```

```
void TexSubImage3D(enum target, int level,
    int xoffset, int yoffset, int zoffset, sizei width,
    sizei height, sizei depth, enum format, enum type,
    const void *data);
```

```
target: TEXTURE_3D, TEXTURE_2D_ARRAY
format, type: See TexImage3D
```

```
void TexSubImage2D(enum target, int level, int xoffset,
    int yoffset, sizei width, sizei height, enum format,
    enum type, const void *data);
```

```
target: TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z}, TEXTURE_2D,
    TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}
format, type: See TexImage3D
```

```
void CopyTexSubImage3D(enum target, int level,
    int xoffset, int yoffset, int zoffset, int x, int y,
    sizei width, sizei height);
```

```
target: TEXTURE_3D, TEXTURE_2D_ARRAY
```

Texturing [3.8]

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: {TEXTURE0..TEXTURE7} where
    i = {MAX_COMBINED_TEXTURE_IMAGE_UNITS-1}
```

```
void GenTextures(sizei n, uint *textures);
```

```
void BindTexture(enum target, uint texture);
```

```
void DeleteTextures(sizei n, const uint *textures);
```

Sampler Objects [3.8.2]

```
void GenSamplers(sizei count, uint *samplers);
```

```
void BindSampler(uint unit, uint sampler);
```

```
void SamplerParameter{if}(uint sampler,
    enum pname, T param);
    pname: TEXTURE_WRAP_{S, T, R}, TEXTURE_{MIN, MAG}_FILTER,
    TEXTURE_{MIN, MAX}_LOD, TEXTURE_COMPARE_{MODE, FUNC}
```

```
void SamplerParameter{if}v(uint sampler, enum pname,
    const T *params);
    pname: See SamplerParameter{if}
```

```
void DeleteSamplers(sizei count, const uint *samplers);
```

Sampler Queries [6.1.5]

```
boolean IsSampler(uint sampler);
```

```
void GetSamplerParameter{if}v(uint sampler,
    enum pname, T *params);
    pname: See SamplerParameter{if}
```

Texture Image Specification [3.8.3, 3.8.4]

```
void TexImage3D(enum target, int level, int internalformat,
    sizei width, sizei height, sizei depth, int border,
    enum format, enum type, const void *data);
```

```
target: TEXTURE_3D, TEXTURE_2D_ARRAY
```

```
format: ALPHA, RGBA, RGB, RG, RED, {RGBA, RGB, RG, RED}_INTEGER,
    DEPTH_{COMPONENT, STENCIL}, LUMINANCE_ALPHA, LUMINANCE
```

(more parameters >)

(Continued on next page >)

Texturing (continued)

void **CopyTexSubImage2D**(enum target, int level, int xoffset, int yoffset, int x, int y, sizei width, sizei height);
 target: TEXTURE_CUBE_MAP_POSITIVE_X, Y, Z, TEXTURE_2D, TEXTURE_CUBE_MAP_NEGATIVE_X, Y, Z

Compressed Texture Images [3.8.6]

void **CompressedTexImage2D**(enum target, int level, enum internalformat, sizei width, sizei height, int border, sizei imageSize, const void *data);
 target: See [TexImage2D](#)
 internalformat: COMPRESSED_RGBA8_ETC2_EAC, COMPRESSED_R11_SIGNED_R11, RG11, SIGNED_RG11, EAC, COMPRESSED_SRGB8_PUNCHTHROUGH_ALPHA1_ETC2, COMPRESSED_SRGB8_ALPHA8_ETC2_EAC [\[Table 3.16\]](#)

void **CompressedTexImage3D**(enum target, int level, enum internalformat, sizei width, sizei height, sizei depth, int border, sizei imageSize, const void *data);
 target: see [TexImage3D](#)
 internalformat: See [TexImage2D](#)

void **CompressedTexSubImage2D**(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, sizei imageSize, const void *data);
 target: See [TexSubImage2D](#)

void **CompressedTexSubImage3D**(enum target, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, sizei imageSize, const void *data);
 target: See [TexSubImage2D](#)

Texture Parameters [3.8.7]

void **TexParameter**(if)(enum target, enum pname, T param);
 void **TexParameter**(if)v(enum target, enum pname, const T *params);
 target: TEXTURE_2D, 3D, TEXTURE_2D_ARRAY, TEXTURE_CUBE_MAP
 pname: TEXTURE_BASE, MAX, LEVEL, TEXTURE_MIN, MAX, LOD, TEXTURE_MIN, MAG, FILTER, TEXTURE_COMPARE_MODE, FUNC, TEXTURE_SWIZZLE_R, G, B, A, TEXTURE_WRAP_S, T, R

Manual Mipmap Generation [3.8.9]

void **GenerateMipmap**(enum target);
 target: TEXTURE_2D, 3D, TEXTURE_2D_ARRAY, CUBE_MAP

Enumerated Queries [6.1.3]

void **GetTexParameter**(if)v(enum target, enum value, T data);
 target: TEXTURE_2D, 3D, TEXTURE_2D_ARRAY, CUBE_MAP
 value: TEXTURE_BASE, MAX, LEVEL, TEXTURE_MIN, MAX, LOD, TEXTURE_MIN, MAG, FILTER, TEXTURE_IMMUTABLE_FORMAT, TEXTURE_COMPARE_FUNC, MODE, TEXTURE_WRAP_S, T, R, TEXTURE_SWIZZLE_R, G, B, A

Texture Queries [6.1.4]

boolean **IsTexture**(uint texture);

Framebuffer Objects**Binding & Managing Framebuffer Objects [4.4.1]**

void **GenFramebuffers**(sizei n, uint *framebuffers);
 void **BindFramebuffer**(enum target, uint framebuffer);
 void **DeleteFramebuffers**(sizei n, const uint *framebuffers);

Renderbuffer Objects [4.4.2]

void **GenRenderbuffers**(sizei n, uint *renderbuffers);
 void **BindRenderbuffer**(enum target, uint renderbuffer);
 target: RENDERBUFFER
 void **DeleteRenderbuffers**(sizei n, const uint *renderbuffers);
 void **RenderbufferStorageMultisample**(enum target, sizei samples, enum internalformat, sizei width, sizei height);
 target: RENDERBUFFER
 internalformat: {R, RG, RGB}8, RGB{565, A4, 5_A1, 10_A2}, RGB{10_A2UI}, R{8, 16, 32}, RG{8, 16, 32}, R{8, 16, 32}UI, RG{8, 16, 32}UI, RGBA, RGBA{8, 8I, 8UI, 16I, 16UI, 32I, 32UI}, SRGB8_ALPHA8, STENCIL_INDEX8, DEPTH{24, 32F}_STENCIL8, DEPTH_COMPONENT{16, 24, 32F}

void **RenderbufferStorage**(enum target, enum internalformat, sizei width, sizei height);
 target: RENDERBUFFER
 internalformat: See [RenderbufferStorageMultisample](#)

Attaching Renderbuffer Images to Framebuffer

void **FramebufferRenderbuffer**(enum target, enum attachment, enum renderbuffertarget, uint renderbuffer);
 (parameters ↗)

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**(float value, boolean invert);

Stencil Test [4.1.4]

Enable/Disable(STENCIL_TEST);
 void **StencilFunc**(enum func, int ref, uint mask);
 func: NEVER, ALWAYS, LESS, GREATER, {L, G}EQUAL, {NOT}EQUAL
 void **StencilFuncSeparate**(enum face, enum func, int ref, uint mask);
 face, func: See [StencilOpSeparate](#)
 void **StencilOp**(enum sfail, enum dpfail, enum dppass);
 sfail, dpfail, and dppass: KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECR_WRAP
 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, GREATER, {L, G}EQUAL, {NOT}EQUAL

Whole Framebuffer Operations**Selecting a Buffer for Writing [4.2.1]**

void **DrawBuffers**(sizei n, const enum *bufs);
 buf's points to an array of n BACK, NONE, or COLOR_ATTACHMENTi where i = [0, MAX_COLOR_ATTACHMENTS - 1].

Fine Control of Buffer Updates [4.2.2]

void **ColorMask**(boolean r, boolean g, 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**(float r, float g, float b, float a);
 void **ClearDepthf**(float d);
 void **ClearStencil**(int s);

Pixel Rectangles [3.7.1]

void **PixelStorei**(enum pname, T param);
 pname: {UN}PACK_ROW_LENGTH, {UN}PACK_ALIGNMENT, {UN}PACK_SKIP_ROWS, PIXELS, {UN}PACK_IMAGE_HEIGHT, {UN}PACK_SKIP_IMAGES
 target: FRAMEBUFFER, {DRAW, READ}_FRAMEBUFFER
 attachment: DEPTH_ATTACHMENT, {DEPTH}_STENCIL_ATTACHMENT, COLOR_ATTACHMENTi (i = [0, MAX_COLOR_ATTACHMENTS-1])
 renderbuffertarget: RENDERBUFFER

Attaching Texture Images to a Framebuffer

void **FramebufferTexture2D**(enum target, enum attachment, enum texture, uint texture, int level);
 texture: TEXTURE_2D, TEXTURE_CUBE_MAP_POSITIVE{X, Y, Z}, TEXTURE_CUBE_MAP_NEGATIVE{X, Y, Z}
 target: FRAMEBUFFER, {DRAW, READ}_FRAMEBUFFER
 attachment: See [FramebufferRenderbuffer](#)
 void **FramebufferTextureLayer**(enum target, enum attachment, uint texture, int level, int layer);
 target: TEXTURE_2D_ARRAY, TEXTURE_3D
 attachment: See [FramebufferRenderbuffer](#)

Framebuffer Completeness [4.4.4]

enum **CheckFramebufferStatus**(enum target);
 target: FRAMEBUFFER, {DRAW, READ}_FRAMEBUFFER
 returns: FRAMEBUFFER_COMPLETE or a constant indicating which value violates framebuffer completeness

Invalidating Framebuffer Contents [4.5]

void **InvalidSubFramebuffer**(enum target, sizei numAttachments, const enum *attachments, int x, int y, sizei width, sizei height);
 target: FRAMEBUFFER
 attachments: points to an array of COLOR, STENCIL, {DEPTH, STENCIL}_ATTACHMENT, COLOR_ATTACHMENTi

Depth Buffer Test [4.1.5]

Enable/Disable(DEPTH_TEST);
 void **DepthFunc**(enum func);
 func: NEVER, ALWAYS, LESS, LEQUAL, EQUAL, GREATER, GEQUAL, NOTEQUAL

Blending [4.1.7]

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, MIN, MAX
 void **BlendFuncSeparate**(enum srcRGB, enum dstRGB, enum srcAlpha, enum dstAlpha);
 srcRGB, dstRGB, srcAlpha, 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_ALPHA_SATURATE
 void **BlendFunc**(enum src, enum dst);
 src, dst: See [BlendFuncSeparate](#)
 void **BlendColor**(float red, float green, float blue, float alpha);

Dithering [4.1.9]

Enable/Disable(DITHER);
 void **ClearBuffer**(if ui)v(enum buffer, int drawbuffer, const T *value);
 buffer: COLOR, DEPTH, STENCIL
 void **ClearBufferfi**(enum buffer, int drawbuffer, float depth, int stencil);
 buffer: DEPTH, STENCIL
 drawbuffer: 0

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 completed.
 void **Flush**(void);
 void **Finish**(void);

Sync Objects and Fences [5.2]

sync **FenceSync**(enum condition, bitfield flags);
 condition: SYNC_GPU_COMMANDS_COMPLETE
 flags: 0
 void **DeleteSync**(sync sync);
 enum **ClientWaitSync**(sync sync, bitfield flags, uint64 timeout);
 flags: 0 or SYNC_FLUSH_COMMANDS_BIT
 timeout: nanoseconds
 void **WaitSync**(sync sync, bitfield flags, uint64 timeout);
 flags: 0
 timeout: TIMEOUT_IGNORED

Hints [5.3]

void **Hint**(enum target, enum hint);
 target: GENERATE_MIPMAP_HINT, FRAGMENT_SHADER_DERIVATIVE_HINT
 hint: FASTEST, NICEST, DONT_CARE

Sync Object Queries [6.1.8]

sync **GetSynciv**(sync sync, enum pname, sizei bufSize, sizei *length, int *values);
 pname: OBJECT_TYPE, SYNC_{STATUS, CONDITION, FLAGS}
 boolean **IsSync**(sync sync);

void **InvalidateFramebuffer**(enum target, sizei numAttachments, const enum *attachments);

Renderbuffer Object Queries [6.1.14]

boolean **IsRenderbuffer**(uint renderbuffer);
 void **GetRenderbufferParameteriv**(enum target, enum pname, int *params);
 target: RENDERBUFFER
 pname: RENDERBUFFER_x, where x may be WIDTH, HEIGHT, {RED, GREEN, BLUE}_SIZE, {ALPHA, DEPTH, STENCIL}_SIZE, SAMPLES, INTERNAL_FORMAT

(Continued on next page >)

Framebuffer Objects (cont'd)

Framebuffer Object Queries [6.1.13]
 boolean **IsFramebuffer**(uint *framebuffer*);
 void
GetFramebufferAttachmentParameteriv(
 enum *target*, enum *attachment*,
 enum *pname*, int **params*);
target: FRAMEBUFFER, {DRAW, READ}_FRAMEBUFFER
attachment: BACK, STENCIL, COLOR_ATTACHMENTi,
 {DEPTH, STENCIL, DEPTH_STENCIL}_ATTACHMENT

(more parameters \rightrightarrows)

pname: FRAMEBUFFER_ATTACHMENT_x,
 where x may be one of OBJECT_{TYPE, NAME},
 COMPONENT_TYPE, COLOR_ENCODING,
 {RED, GREEN, BLUE, ALPHA}_SIZE,
 {DEPTH, STENCIL}_SIZE, TEXTURE_{LEVEL, LAYER},
 TEXTURE_CUBE_MAP_FACE

void **GetInternalformativ**(enum *target*,
 enum *internalformat*, enum *pname*,
 sizei *bufSize*, int **params*);
internalformat:
 See **RenderbufferStorageMultisample**
target: RENDERBUFFER
pname: NUM_SAMPLE_COUNTS, SAMPLES

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 *pname*,
 boolean **data*);
 void **GetIntegerv**(enum *pname*, int **data*);
 void **GetInteger64v**(enum *pname*,
 int64 **data*);
 void **GetFloatv**(enum *pname*, float **data*);

void **GetIntegeri_v**(enum *target*,
 uint *index*, int **data*);
 void **GetInteger64i_v**(enum *target*,
 uint *index*, int64 **data*);
 boolean **IsEnabled**(enum *cap*);

String Queries [6.1.6]
 ubyte ***GetString**(enum *name*);
name: VENDOR, RENDERER, EXTENSIONS,
 {SHADING_LANGUAGE}_VERSION
 ubyte ***GetStringi**(enum *name*, uint *index*);
name: EXTENSIONS

OpenGL ES Shading Language 3.0 Reference Card

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] 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 <i>type-name</i> { <i>members</i> } <i>struct-name</i> []; // optional variable declaration, // optionally an array
Arrays	float <i>foo</i> [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        #ifndef     #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.

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 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.xy

{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

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	linkage into a shader from a previous stage
centroid in	linkage into a shader from a previous stage
out	linkage out of a shader to a subsequent stage
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 precede 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

(Continued on next page >)

Qualifiers (continued)

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

	FP Range	FP Magnitude Range	FP Precision	Integer Range	
				Signed	Unsigned
highp	$(-2^{25}, 2^{27})$	$0.0, (2^{-26}, 2^{27})$	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

Aggregate Operations and Constructors

Matrix Constructor Examples [5.4.2]

```
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
```

(more examples ↗)

Statements and Structure

Iteration and Jumps [6]

Iteration	for (;) { break, continue } while () { break, continue } do { break, continue } while ();
Selection	if () { } if () { } else { } switch () { break, case }
Jump	break, continue, return discard // Fragment shader only
Entry	void main()

```
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. Valid operators are:

.	field selector
==	equality
=	assignment

Array Operations [5.7]

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

```
diffuseColor += lightIntensity[3] * NdotL;
```

The size of an array can be determined using the .length() operator. For example:

```
for (i = 0; i < a.length(); i++)
    a[i] = 0.0;
```

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]

Inputs:

```
int gl_VertexID; // integer index
int gl_InstanceID; // instance number
```

Outputs:

```
out gl_PerVertex {
    vec4 gl_Position; // transformed vertex position in clip coordinates
    float gl_PointSize; // transformed point size in pixels (point rasterization only)
};
```

Fragment Shader Special Variables [7.2]

Inputs:

```
highp vec4 gl_FragCoord; // fragment position within frame buffer
bool gl_FrontFacing; // fragment belongs to a front-facing primitive
mediump vec2 gl_PointCoord; // 0.0 to 1.0 for each component
```

Outputs:

```
highp float gl_FragDepth; // depth range
```

Built-In Constants With Minimum Values [7.3]

Built-in Constant	Minimum value
const mediump int gl_MaxVertexAttribs	16
const mediump int gl_MaxVertexUniformVectors	256
const mediump int gl_MaxVertexOutputVectors	16
const mediump int gl_MaxFragmentInputVectors	15
const mediump int gl_MaxVertexTextureImageUnits	16
const mediump int gl_MaxCombinedTextureImageUnits	32
const mediump int gl_MaxTextureImageUnits	16
const mediump int gl_MaxFragmentUniformVectors	224
const mediump int gl_MaxDrawBuffers	4
const mediump int gl_MinProgramTexelOffset	-8
const mediump int gl_MaxProgramTexelOffset	7

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.

```
struct gl_DepthRangeParameters {
    float near; // n
    float far; // f
    float diff; // f - n
};
uniform gl_DepthRangeParameters gl_DepthRange;
```

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
T sin (T angle);	sine
T cos (T angle);	cosine
T tan (T angle);	tangent
T asin (T x);	arc sine

(more angle & trigonometry functions ↗)

Angle & Trigonometry Functions (continued)

T acos (T x);	arc cosine
T atan (T y, T x); T atan (T y_over_x);	arc tangent
T sinh (T x);	hyperbolic sine
T cosh (T x);	hyperbolic cosine
T tanh (T x);	hyperbolic tangent
T asinh (T x);	arc hyperbolic sine; inverse of sinh
T acosh (T x);	arc hyperbolic cosine; non-negative inverse of cosh
T atanh (T x);	arc hyperbolic tangent; inverse of tanh

Exponential Functions [8.2]

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

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

(Continued on next page)

Built-In Functions (continued)

Common Functions [8.3]

Component-wise operation. T is float and *vecn*, TI is int and *ivec*, TU is uint and *uvecn*, and TB is bool and *bvecn*, where n is 2, 3, or 4.

T abs (T x); TI abs (TI x);	absolute value
T sign (T x); TI sign (TI x);	returns -1.0, 0.0, or 1.0
T floor (T x);	nearest integer <= x
T trunc (T x);	nearest integer a such that a <= x
T round (T x);	round to nearest integer
T roundEven (T x);	round to nearest integer
T ceil (T x);	nearest integer >= x
T fract (T x);	x - floor (x)
T mod (T x, T y); T mod (T x, float y); T modf (T x, out T i);	modulus
T min (T x, T y); TI min (TI x, TI y); TU min (TU x, TU y); T min (T x, float y); TI min (TI x, int y); TU min (TU x, uint y);	minimum value
T max (T x, T y); TI max (TI x, TI y); TU max (TU x, TU y); T max (T x, float y); TI max (TI x, int y); TU max (TU x, uint y);	maximum value
T clamp (T i x, T <i>minVal</i> , T <i>maxVal</i>); TI clamp (V x, TI <i>minVal</i> , TI <i>maxVal</i>); TU clamp (TU x, TU <i>minVal</i> , TU <i>maxVal</i>); T clamp (T x, float <i>minVal</i> , float <i>maxVal</i>); TI clamp (TI x, int <i>minVal</i> , int <i>maxVal</i>); TU clamp (TU x, uint <i>minVal</i> , uint <i>maxVal</i>);	min (<i>max</i> (x, <i>minVal</i>), <i>maxVal</i>)
T mix (T x, T y, T a); T mix (T x, T y, float a);	linear blend of x and y
T mix (T x, T y, TB a);	Selects vector source for each returned component
T step (T <i>edge</i> , T x); T step (float <i>edge</i> , T x);	0.0 if x < <i>edge</i> , else 1.0
T smoothstep (T <i>edge0</i> , T <i>edge1</i> , T x); T smoothstep (float <i>edge0</i> , float <i>edge1</i> , T x);	clamp and smooth
TB isnan (T x);	true if x is a NaN
TB isinf (T x);	true if x is positive or negative infinity
TI floatBitsToInt (T <i>value</i>); TU floatBitsToUint (T <i>value</i>);	highp integer, preserving float bit level representation
T intBitsToFloat (TI <i>value</i>); TU uintBitsToFloat (TU <i>value</i>);	highp float, preserving integer bit level representation

Floating-Point Pack and Unpack Functions [8.4]

uint packSnorm2x16 (vec2 v); uint packUnorm2x16 (vec2 v);	convert two floats to fixed point and pack into an integer
vec2 unpackSnorm2x16 (uint p); vec2 unpackUnorm2x16 (uint p);	unpack fixed point value pair into floats
uint packHalf2x16 (vec2 v);	convert two floats into half-precision floats and pack into an integer
vec2 unpackHalf2x16 (uint v);	unpack half value pair into full floats

Geometric Functions [8.5]

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

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

(more Geometric Functions ↗)

Geometric Functions (continued)

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 <i>eta</i>);	refraction vector

Matrix Functions [8.6]

Type mat is any matrix type.

mat matrixCompMult (mat x, mat y);	multiply x by y component-wise
mat2 outerProduct (vec2 c, vec2 r); mat3 outerProduct (vec3 c, vec3 r); mat4 outerProduct (vec4 c, vec4 r);	linear algebraic column vector * row vector
mat2x3 outerProduct (vec3 c, vec2 r); mat3x2 outerProduct (vec2 c, vec3 r); mat2x4 outerProduct (vec4 c, vec2 r); mat4x2 outerProduct (vec2 c, vec4 r); mat3x4 outerProduct (vec4 c, vec3 r); mat4x3 outerProduct (vec3 c, vec4 r);	linear algebraic column vector * row vector
mat2 transpose (mat2 m); mat3 transpose (mat3 m); mat4 transpose (mat4 m); mat2x3 transpose (mat3x2 m); mat3x2 transpose (mat2x3 m); mat2x4 transpose (mat4x2 m); mat4x2 transpose (mat2x4 m); mat3x4 transpose (mat4x3 m); mat4x3 transpose (mat3x4 m);	transpose of matrix m
float determinant (mat2 m); float determinant (mat3 m); float determinant (mat4 m);	determinant of matrix m
mat2 inverse (mat2 m); mat3 inverse (mat3 m); mat4 inverse (mat4 m);	inverse of matrix m

Vector Relational Functions [8.7]

Compare x and y component-wise. Input and return vector sizes for a particular call must match. Type *bvec* is *bvecn*; vec is *vecn*; *ivec* is *ivec*; *uvec* is *uvecn*; (where n is 2, 3, or 4). T is union of *vec* and *ivec*.

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

Texture Lookup Functions [8.8]

The function *textureSize* returns the dimensions of level *lod* for the texture bound to *sampler*, as described in [2.11.9] of the OpenGL ES 3.0 specification, under "Texture Size Query". The initial "g" in a type name is a placeholder for nothing, "i", or "u".

highp <i>ivec</i> {2,3} textureSize (<i>gsampler</i> {2,3}D <i>sampler</i> , int <i>lod</i>); highp <i>ivec</i> 2 textureSize (<i>gsampler</i> Cube <i>sampler</i> , int <i>lod</i>); highp <i>ivec</i> 2 textureSize (<i>sampler</i> 2DShadow <i>sampler</i> , int <i>lod</i>); highp <i>ivec</i> 2 textureSize (<i>sampler</i> CubeShadow <i>sampler</i> , int <i>lod</i>); highp <i>ivec</i> 3 textureSize (<i>gsampler</i> 2DArray <i>sampler</i> , int <i>lod</i>); highp <i>ivec</i> 3 textureSize (<i>sampler</i> 2DArrayShadow <i>sampler</i> , int <i>lod</i>);	
---	--

Texture lookup functions using samplers are available to vertex and fragment shaders. The initial "g" in a type name is a placeholder for nothing, "i", or "u".

<i>gvec</i> 4 texture (<i>gsampler</i> {2,3}D <i>sampler</i> , <i>vec</i> {2,3} P [, float <i>bias</i>]);	
<i>gvec</i> 4 texture (<i>gsampler</i> Cube <i>sampler</i> , <i>vec</i> 3 P [, float <i>bias</i>]);	
float texture (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 3 P [, float <i>bias</i>]);	
float texture (<i>sampler</i> CubeShadow <i>sampler</i> , <i>vec</i> 4 P [, float <i>bias</i>]);	
<i>gvec</i> 4 texture (<i>gsampler</i> 2DArray <i>sampler</i> , <i>vec</i> 3 P [, float <i>bias</i>]);	
float texture (<i>sampler</i> 2DArrayShadow <i>sampler</i> , <i>vec</i> 4 P);	

(more Texture Lookup functions ↗)

Texture Lookup Functions (continued)

<i>gvec</i> 4 textureProj (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> {3,4} P [, float <i>bias</i>]);	
<i>gvec</i> 4 textureProj (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 4 P [, float <i>bias</i>]);	
float textureProj (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 4 P [, float <i>bias</i>]);	
<i>gvec</i> 4 textureLod (<i>gsampler</i> {2,3}D <i>sampler</i> , <i>vec</i> {2,3} P, float <i>lod</i>);	
<i>gvec</i> 4 textureLod (<i>gsampler</i> Cube <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i>);	
float textureLod (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i>);	
<i>gvec</i> 4 textureLod (<i>gsampler</i> 2DArray <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i>);	
<i>gvec</i> 4 textureOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 2 P, <i>ivec</i> 2 <i>offset</i> [, float <i>bias</i>]);	
<i>gvec</i> 4 textureOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 3 P, <i>ivec</i> 3 <i>offset</i> [, float <i>bias</i>]);	
float textureOffset (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 3 P, <i>ivec</i> 2 <i>offset</i> [, float <i>bias</i>]);	
<i>gvec</i> 4 textureOffset (<i>gsampler</i> 2DArray <i>sampler</i> , <i>vec</i> 3 P, <i>ivec</i> 2 <i>offset</i> [, float <i>bias</i>]);	
<i>gvec</i> 4 texelFetch (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 2 P, int <i>lod</i>);	
<i>gvec</i> 4 texelFetch (<i>gsampler</i> 3D <i>sampler</i> , <i>ivec</i> 3 P, int <i>lod</i>);	
<i>gvec</i> 4 texelFetch (<i>gsampler</i> 2DArray <i>sampler</i> , <i>ivec</i> 3 P, int <i>lod</i>);	
<i>gvec</i> 4 texelFetchOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>ivec</i> 2 P, int <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 texelFetchOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>ivec</i> 3 P, int <i>lod</i> , <i>ivec</i> 3 <i>offset</i>);	
<i>gvec</i> 4 texelFetchOffset (<i>gsampler</i> 2DArray <i>sampler</i> , <i>ivec</i> 3 P, int <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 3 P, <i>ivec</i> 2 <i>offset</i> [, float <i>bias</i>]);	
<i>gvec</i> 4 textureProjOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 4 P, <i>ivec</i> 3 <i>offset</i> [, float <i>bias</i>]);	
float textureProjOffset (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 4 P, <i>ivec</i> 2 <i>offset</i> [, float <i>bias</i>]);	
<i>gvec</i> 4 textureLodOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 2 P, float <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureLodOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i> , <i>ivec</i> 3 <i>offset</i>);	
float textureLodOffset (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureLodOffset (<i>gsampler</i> 2DArray <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjLod (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i>);	
<i>gvec</i> 4 textureProjLod (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 4 P, float <i>lod</i>);	
<i>gvec</i> 4 textureProjLod (<i>sampler</i> 3D <i>sampler</i> , <i>vec</i> 4 P, float <i>lod</i>);	
float textureProjLod (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 4 P, float <i>lod</i>);	
<i>gvec</i> 4 textureProjLodOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 3 P, float <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjLodOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 4 P, float <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjLodOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 4 P, float <i>lod</i> , <i>ivec</i> 3 <i>offset</i>);	
float textureProjLodOffset (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 4 P, float <i>lod</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureGrad (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 2 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
<i>gvec</i> 4 textureGrad (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 3 <i>dPdx</i> , <i>vec</i> 3 <i>dPdy</i>);	
<i>gvec</i> 4 textureGrad (<i>gsampler</i> Cube <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 3 <i>dPdx</i> , <i>vec</i> 3 <i>dPdy</i>);	
float textureGrad (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
float textureGrad (<i>sampler</i> CubeShadow <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 3 <i>dPdx</i> , <i>vec</i> 3 <i>dPdy</i>);	
<i>gvec</i> 4 textureGrad (<i>gsampler</i> 2DArray <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
float textureGrad (<i>sampler</i> 2DArrayShadow <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
<i>gvec</i> 4 textureGradOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 2 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureGradOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 3 <i>dPdx</i> , <i>vec</i> 3 <i>dPdy</i> , <i>ivec</i> 3 <i>offset</i>);	
float textureGradOffset (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureGradOffset (<i>gsampler</i> 2DArray <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	
float textureGradOffset (<i>sampler</i> 2DArrayShadow <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjGrad (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
<i>gvec</i> 4 textureProjGrad (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
<i>gvec</i> 4 textureProjGrad (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 3 <i>dPdx</i> , <i>vec</i> 3 <i>dPdy</i>);	
float textureProjGrad (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i>);	
<i>gvec</i> 4 textureProjGradOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 3 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjGradOffset (<i>gsampler</i> 2D <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	
<i>gvec</i> 4 textureProjGradOffset (<i>gsampler</i> 3D <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 3 <i>dPdx</i> , <i>vec</i> 3 <i>dPdy</i> , <i>ivec</i> 3 <i>offset</i>);	
float textureProjGradOffset (<i>sampler</i> 2DShadow <i>sampler</i> , <i>vec</i> 4 P, <i>vec</i> 2 <i>dPdx</i> , <i>vec</i> 2 <i>dPdy</i> , <i>ivec</i> 2 <i>offset</i>);	

Fragment Processing Functions [8.9]

Approximated using local differencing.

T dFdx (T p);	Derivative in x
T dFdy (T p);	Derivative in y
T fwidth (T p);	abs (dFdx (p)) + abs (dFdy (p));



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.