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] refers to sections in the OpenGL ES Shading Language 1.0 specification.

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

OpenGL ES Command Syntax [2.3]

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

```
return-type Name[1234](if){(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, GType

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 Buffers Objects

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

Creating Buffer Object Data Stores

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

Texture [3.7]

Shaders support textureing using at least MAX_TEXTURE_IMAGE_UNITS image slots for vertex shaders and at least MAX_TEXTURE_IMAGE_UNITS image slots for fragment shaders.

void ActiveTexture(enum target);
texture: [TEXTURE0_TEXTURE1], where i = MAX_COMBINED_TEXTURE_IMAGE_UNITS - 1

Texture Image Specification [3.7.1]

void TexImage2D(enum target, int level, int internalformat, size width, size height, int border, enum format, enum type, void *data);
target: TEXTURE_2D, TEXTURE_CUBE_MAP
internalformat: ALPHA, LUMINANCE, ALPHA, RGB, RGBA
format: ALPHA, RGB, RGBA, LUMINANCE, LUMINANCE ALPHA, RGBA

void TexSubImage2D(enum target, int level, int xoffset, int yoffset, size width, size height, int border, enum format, enum type, void *data);
target: TEXTURE_2D, TEXTURE_CUBE_MAP
internalformat: ALPHA, LUMINANCE, ALPHA, RGB, RGBA

void CopyTexSubImage2D(enum target, int level, int xoffset, int yoffset, size width, size height, int border, enum format, enum type, void *data);
target: TEXTURE_2D, TEXTURE_CUBE_MAP
internalformat: ALPHA, LUMINANCE, ALPHA, RGB, RGBA

Conversion from RGBA pixel components to internal texture components:

<table>
<thead>
<tr>
<th>Basic Internal Format</th>
<th>RGBA</th>
<th>Internal Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHA</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>LUMINANCE</td>
<td>R</td>
<td>L</td>
</tr>
<tr>
<td>LUMINANCE ALPHA</td>
<td>A</td>
<td>A, L</td>
</tr>
<tr>
<td>RGB</td>
<td>R, G, B</td>
<td>R, G, B</td>
</tr>
<tr>
<td>RGBA</td>
<td>R, G, B, A</td>
<td>R, G, B, A</td>
</tr>
</tbody>
</table>

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, int internalformat, int x, int y, size width, size height, int border);
target: TEXTURE_2D, TEXTURE_CUBE_MAP
internalformat: ALPHA, LUMINANCE, RGBA

void TexSubImage2D(enum target, int level, int xoffset, int yoffset, size width, size height, int border, enum format, enum type, void *data);
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 GetTexParameterf(enum target, enum name, T param);
target: TEXTURE_2D, TEXTURE_CUBE_MAP

Texture Queries [6.1.4]

boolean IsTexture(uint texture);

Errors [2.5]

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

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]

GL types are not C types.

<table>
<thead>
<tr>
<th>GL Type</th>
<th>Minimum Bit Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
<td>Boolean</td>
</tr>
<tr>
<td>byte</td>
<td>8</td>
<td>Signed binary integer</td>
</tr>
<tr>
<td>ubyte</td>
<td>8</td>
<td>Unsigned binary integer</td>
</tr>
<tr>
<td>char</td>
<td>8</td>
<td>Characters making up strings</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>Signed 2's complement binary integer</td>
</tr>
<tr>
<td>ushort</td>
<td>16</td>
<td>Unsigned binary integer</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>Signed 2's complement binary integer</td>
</tr>
<tr>
<td>uint</td>
<td>32</td>
<td>Unsigned binary integer</td>
</tr>
<tr>
<td>fixed</td>
<td>32</td>
<td>Signed 2's complement 16.16 scaled integer</td>
</tr>
<tr>
<td>size</td>
<td>32</td>
<td>Non-negative binary integer</td>
</tr>
<tr>
<td>enum</td>
<td>32</td>
<td>Enumerated binary integer</td>
</tr>
<tr>
<td>intptr</td>
<td>32</td>
<td>Signed 2's complement binary integer</td>
</tr>
<tr>
<td>sizeptr</td>
<td>32</td>
<td>Non-negative binary integer</td>
</tr>
<tr>
<td>bitfield</td>
<td>32</td>
<td>Bit field</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>Floating-point value</td>
</tr>
<tr>
<td>clampf</td>
<td>32</td>
<td>Floating-point value cramped to [0; 1]</td>
</tr>
</tbody>
</table>

Vertices Current Vertex State [2.7]

void VertexAttrib[1234](f[uint index, index, enum type, boolean normalized, sizei stride, const void *pointer], type SIDE, ENUM gl type, enum type, void *data);
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 buffer bound, and pointer is treated as an offset within the buffer.

void EnableVertexAttribArray[uint index]

void DisableVertexAttribArray[uint index]

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

void EnableArrayBuf[STATIC_DRAW, DYNAMIC_DRAW]

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

Rasterization [3]

Points [3.3]

Point size is taken from the shader built-in 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];
dir: CCW, CW
void CullFace[enum mode];
mode: FRONT, BACK, FRONT_AND_BACK
Enable/Disable[CULL_FACE]

void PolygonOffset[static factor, float units];
Enable/Disable[POLYGON_OFFSET_FILL]

Pixel Rectangles [3.6, 4.3]

void PixelStore[enum name, int param];
name: UNPACK_ALIGNMENT, PACK_ALIGNMENT
Shaders and Programs

Shaders Object [2.10.1]
uint CreateShader(enum type);
type: VERTEX, FRAGMENT, SHADER
void ShaderSource(char *source, count *source);
void CompileShader(uint shader);
void ReleaseShaderCompiler(void);
void DeleteShader(uint shader);

Loading Shader Binaries [2.10.2]
void ShaderBinary(char *source, count *source);
void CreateProgram(uint void);
count ShaderBinary(void);
void ShaderSource(char *source, count *source);
void CreateShader(uint void);

Shader Variables [2.10.4]
Vertex Attributes
gGetVertexAttrib(name, count, int *shaders, enum binaryFormat, count *binary, size *length);

Program Objects [2.10.3]
count ProgramBinary(void);
void AttachShader(uint program, uint shader);
void LinkProgram(uint program);
void UseProgram(uint program);
void DeleteProgram(uint program);

Program Queries [6.1.8]

Shader Queries [6.1.8]
bool IsShader(name);
gGetShaderInfo(name, count, int *params);

Depth Buffer Test [4.1.5]
Enable/Disable(DEPTH_TEST) depthFunction(name);
func: NEVER, ALWAYS, LESS, EQUAL, GREATER, LESS_EQUAL, GREATER_EQUAL, NOTEQUAL

Blending [4.1.6]
Enable/Disable(BLEND)
blendEquation(mode, modeRGB, modeAlpha);
blendEquationSeparate(enum modeRGB, enum modeAlpha);
blendSeparate(name srcRGB, enum dstRGB, name srcAlpha, enum dstAlpha);
blend(name srcRGB, enum dstRGB, dstAlpha, ZERO, ONE, ONE_MINUS_SRC_COLOR, srcRGB, srcAlpha, srcRGB, srcRG);
blendfunc(name srcRGB, enum dstRGB, dstAlpha, ZERO, ONE, ONE_MINUS_SRC_COLOR, srcRGB, srcAlpha, srcRGB, srcRG);
blendfuncSeparate(name srcRGB, enum dstRGB, dstAlpha, name srcAlpha, enum dstAlpha);
blendSeparate(name srcRGB, enum dstRGB, dstAlpha, name srcAlpha, enum dstAlpha);
blendfuncSeparate(name srcRGB, enum dstRGB, dstAlpha, name srcAlpha, enum dstAlpha);
blendSeparate(name srcRGB, enum dstRGB, dstAlpha, name srcAlpha, enum dstAlpha);

Dithering [4.1.7]
Enable/Disable(DITHER)

Clearing the Buffers [4.2.3]
void Clear(name buffer);
buf: BUFFER_BIT, NAME_BUFFER_BIT, DEPTH_BUFFER_BIT, STENCIL_BUFFER_BIT;
name: CLEAR_BUFFER_BIT, CLEAR_DEPTH_BUFFER_BIT, CLEAR_STENCIL_BUFFER_BIT;

color: NEVER, ALWAYS, LESS, EQUAL, GREATER, LESS_EQUAL, GREATER_EQUAL, NOTEQUAL

Framebuffer Objects

Binding & Managing Framebuffer Objects [4.4.1]
gBindFramebuffer(target, name framebuffer);

target: FRAMEBUFFER

gDeleteFramebuffers(size, int *framebuffers);
gGenFramebuffers(size, int *framebuffers);

Renderer Objects [4.4.2]
bool BindRenderer(target, name renderer);
target: RENDERERBUFFER

gDeleteRenderbuffers(size, int *renderbuffers);
gGenRenderbuffers(size, int *renderbuffers);

gBufferStorage(name target, enum internalFormat, size *length *size);
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
- i vec2, i vec3, i vec4: 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: // optional variable declaration,
- } // optionally an array

Arrays
- float foo[3];
- 

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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>parenthetical grouping</td>
<td>N/A</td>
</tr>
<tr>
<td>[]</td>
<td>array subscript</td>
<td>L - R</td>
</tr>
<tr>
<td>++ --</td>
<td>prefix increment and decrement</td>
<td>L - R</td>
</tr>
<tr>
<td>* /</td>
<td>multiplicative</td>
<td>L - R</td>
</tr>
<tr>
<td>+ -</td>
<td>additive</td>
<td>L - R</td>
</tr>
<tr>
<td>&gt; &lt; $&lt;=$</td>
<td>relational</td>
<td>L - R</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>logical and</td>
<td>L - R</td>
</tr>
<tr>
<td>^</td>
<td>logical exclusive or</td>
<td>L - R</td>
</tr>
<tr>
<td></td>
<td>logical inclusive or</td>
<td>L - R</td>
</tr>
<tr>
<td>?:</td>
<td>selection (Selects one entire operand. Use mix() to select individual components of vectors.)</td>
<td>L - R</td>
</tr>
<tr>
<td>+= -=</td>
<td>assignment</td>
<td>L - R</td>
</tr>
<tr>
<td>,</td>
<td>sequence</td>
<td>L - R</td>
</tr>
</tbody>
</table>

Vector Components [5.5]
In addition to array numeric subcript syntax, names of vector components are denoted by a single letter. Components can be swizzled and replicated, e.g.: pos.xx, equal(a), etc.

- \(a, b, x, y\) Use when accessing vectors that represent points or normals
- \(r, g, b\) Use when accessing vectors that represent colors
- \(a, b, t, q\) 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

Attribute
- attribute: Linkage between a vertex shader and OpenGL ES for per-vertex 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 with any basic data types, to declare a variable whose type is a structure, or an array of any of these. For example:

```glsl
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:

```glsl
varying vec3 normal;
```

Parameter Qualifiers [4.3]
Input values are copied at function call time, output values are copied out after function return time.

- none (Default) same as uniform
- in
- out
- inout

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:

```glsl
lowp float color;
```

Order of Qualification [4.7]
When multiple qualifications are present, they must follow a strict order. This order is as follows:

- invariable, storage, precision
- parameter, qualifier
- uniform, varying
- highp, mediump, lowp

Aggregate Operations and Constructors
Matrix Constructor Examples [5.4]

```glsl
mat2(float); // init diagonal
mat2(vec2, vec2); // column-major order
mat2(float, float, float, float); // column-major order
```

Structure Constructor Example [5.4.3]

```glsl
struct light { float lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

Matrix Components [5.6]
Access components of a matrix with array subscript syntax.

```glsl
mat4 m;
```

Examples of operations on matrices and vectors:

```glsl
m = f * m; // scalar * matrix component-wise
v = f * v; // scalar * vector component-wise
```

Array Operations [4.1.9]
Array elements are accessed using the array subscript operator "[]". For example:

```glsl
diffuseColor += lightIntensity[3] * NdotL;
```
Built-In Functions [8.1]
Component-wise operation. Parameters specified as angle are assumed to be in units of radians. T is float, vec2, vec3, vec4.

- \( \text{radians} \) degrees to radians
- \( \text{degrees} \) radians to degrees
- \( \sin(\text{angle}) \) sine
- \( \cos(\text{angle}) \) cosine
- \( \tan(\text{angle}) \) tangent
- \( \text{asin}(x) \) arc sine
- \( \text{acos}(x) \) arc cosine
- \( \text{atan}(x, y) \) arc tangent
- \( \text{atan}(x, y_{\text{over}_x}) \) linear blend of \( y \), \( x \), \( a \)
- \( \text{clamp}(x, y, z) \) minimum value \( x \), \( y \), \( z \)
- \( \text{smoothstep}(x, y, z) \) 0.0 if \( x < \text{edge} \), else 1.0
- \( \text{smoothstep}(x_{\text{edge}1}, x_{\text{edge}2}, y) \) clip and smooth
- \( \text{matrixCompMult} \) dot product
- \( \text{distance} \) distance between points
- \( \text{dot} \) dot product
- \( \text{cross} \) cross product
- \( \text{length} \) length of vector
- \( \text{normalize} \) normalize vector to length 1
- \( \text{reflect} \) reflection direction \( \text{I} \) = \( \text{N} \) \( \times \) \( \eta \)
- \( \text{refract} \) refraction vector
- \( \text{cross} \) cross product
- \( \text{notEqual} \) true if any component of \( x \) are true
- \( \text{allEqual} \) true if all components of \( x \) are true
- \( \text{bool}(x) \) logical complement of \( x \)
- \( \text{equal}(x, y) \) true if all components of \( x \) are equal
- \( \text{near(x)} \) nearest integer \( \leq x \)
- \( \text{sqrt} \) nearest integer \( \geq x \)
- \( \text{abs}(x) \) absolute value
- \( \text{sign}(x) \) returns -1.0, 0.0, or 1.0
- \( \text{fract}(x) \) \( x \) \( \times \) \( \text{floor}(x) \)
- \( \text{mod}(x, y) \) modulus
- \( \text{min}(x, y) \) minimum value
- \( \text{max}(x, y) \) maximum value
- \( \text{mix}(x, y, a) \) linear blend of \( x \) and \( y \)
- \( \text{step}(x, y) \) 0.0 if \( x < \text{edge} \), else 1.0
- \( \text{smoothstep}(x_{\text{edge}1}, x_{\text{edge}2}, y) \) clip and smooth

Geometric Functions [8.4]
These functions operate on vectors as vectors, not component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is bvec2; vec is vec2, vec3, vec4.

- \( \text{floatLength}(x) \) length of vector
- \( \text{floatDistance}(x, y, z) \) distance between points
- \( \text{vec3Cross}(x, y, z) \) cross product
- \( \text{normalize}(x) \) normalize vector to length 1
- \( \text{max}(x, y) \) maximum value \( x \), \( y \)
- \( \text{min}(x, y) \) minimum value \( x \), \( y \)
- \( \text{atan2}(x, y) \) arc tangent

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

- \( \text{exp}(x) \) \( e^x \)
- \( \text{exp2}(x) \) \( 2^x \)
- \( \text{log}(x) \) \( \ln(x) \)
- \( \text{log2}(x) \) \( \log_2(x) \)
- \( \text{sqrt}(x) \) \( \sqrt{x} \)
- \( \text{inverse}(x) \) \( 1/x \)

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

- \( \text{abs}(x) \) absolute value
- \( \text{sign}(x) \) returns -1.0, 0.0, or 1.0
- \( \text{floor}(x) \) nearest integer \( \leq x \)
- \( \text{ceil}(x) \) nearest integer \( \geq x \)
- \( \text{fract}(x) \) \( x \) \( \times \) \( \text{floor}(x) \)
- \( \text{mod}(x, y) \) modulus
- \( \text{min}(x, y) \) minimum value
- \( \text{max}(x, y) \) maximum value
- \( \text{mix}(x, y, a) \) linear blend of \( x \) and \( y \)
- \( \text{step}(x, y) \) 0.0 if \( x < \text{edge} \), else 1.0
- \( \text{smoothstep}(x_{\text{edge}1}, x_{\text{edge}2}, y) \) clip and smooth

Vector Relational Functions [8.6]
Component-wise comparison operations. T is float, vec2, vec3, vec4.

- \( \text{bvecLessThan}(x, y) \) \( x \) \( < \) \( y \)
- \( \text{bvecLessThanEqual}(x, y) \) \( x \) \( \leq \) \( y \)
- \( \text{bvecGreaterThan}(x, y) \) \( x \) \( > \) \( y \)
- \( \text{bvecGreaterThanEqual}(x, y) \) \( x \) \( \geq \) \( y \)
- \( \text{bvecEqual}(x, y) \) \( x \) \( = \) \( y \)
- \( \text{bvecNotequal}(x, y) \) \( x \) \( \neq \) \( y \)
- \( \text{bvecAllEqual}(x, y) \) \text{true if all components of } x \text{ are true}
- \( \text{bvecAllNotequal}(x, y) \) \text{true if all components of } x \text{ are true}
- \( \text{bool}(x) \) \text{logical complement of } x

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

- \( \text{vec4(texture2Dlod}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}

Available only in fragment shaders.

- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}

Available in vertex and fragment shaders.

- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}
- \( \text{vec4(texture2DlodSampler2D}(x, y) \text{lod}) \) \text{texture lookup at } \text{lod}

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