OpenGL® ES
Common/Common-Lite Profile Specification

Version 1.1.11 (Difference Specification) (Annotated)

Editor (version 1.0): David Blythe
Editor (version 1.1): Aaftab Munshi

April 4, 2007

Contents

1 Overview **1**
1.1 Conventions . . . 1

2 OpenGL Operation **3**
2.1 OpenGL Fundamentals . . . 3
2.1.1 Fixed-Point Computation . . . 4
2.2 GL State . . . 4
2.3 GL Command Syntax . . . 4
2.4 Basic GL Operation . . . 5
2.5 GL Errors . . . 5
2.6 Begin/End Paradigm . . . 5
2.7 Vertex Specification . . . 6
2.8 Vertex Arrays . . . 7
2.9 Buffer Objects . . . 8
2.10 Rectangles . . . 9
2.11 Coordinate Transformations . . . 9
2.12 Clipping . . . 11
2.13 Current Raster Position . . . 11
2.14 Colors and Coloring . . . 11

3 Rasterization **13**
3.1 Invariance . . . 13
3.2 Antialiasing . . . 13
3.3 Points . . . 13
3.4 Line Segments . . . 14
3.5 Polygons . . . 14
3.6 Pixel Rectangles . . . 15
3.7 Bitmaps . . . 17
3.8 Texturing . . . 17
3.8.1 Copy Texture . . . 17
3.8.2 Compressed Textures . . . 18
3.8.3 Texture Addressing Modes . . . 18
3.8.4 Texture Completeness . . . 18
3.8.5 Texture State . . . 19
3.8.6 Texture Environments and Texture Functions . . . 22
3.9 Fog . . . 25

4 Per-Fragment Operations and the Framebuffer **26**
4.1 Per-Fragment Operations . . . 26
4.1.1 Blending . . . 28
4.2 Whole Framebuffer Operations . . . 29
4.3 Drawing, Reading, and Copying Pixels . . . 29

5 Special Functions **31**
5.1 Evaluators . . . 31
5.2 Selection . . . 31
5.3 Feedback . . . 32
5.4 Display Lists . . . 32
5.5 Flush and Finish . . . 32
5.6 Hints . . . 33

6 State and State Requests **34**
6.1 Querying GL State . . . 34
6.2 State Tables . . . 36

7 Core Additions and Extensions **53**
7.1 Byte Coordinates . . . 54
7.2 Fixed Point . . . 54
7.3 Single-precision Commands . . . 55
7.4 Compressed Paletted Texture . . . 56
7.5 Read Format . . . 56
7.6 Matrix Palette . . . 56
7.7 Point Sprites . . . 56
7.8 Point Size Array . . . 56
7.9 Matrix Get . . . 57
7.10 Draw Texture . . . 57

8 Packaging **58**
8.1 Header Files . . . 58
8.2 Libraries . . . 58

A Acknowledgements **59**

B OES Extension Specifications **61**
B.1 OES_byte_coordinates . . . 61
B.2 OES_fixed_point . . . 62
B.3 OES_single_precision . . . 63
B.4 OES_read_format . . . 64
B.5 OES_query_matrix . . . 65
B.6 OES_compressed_paletted_texture . . . 66
B.7 OES_matrix_palette . . . 67
B.8 OES_point_sprite . . . 68
B.9 OES_point_size_array . . . 69
B.10 OES_matrix_get . . . 70

B.11 OES_draw_texture 71

Chapter 1

Overview

This document outlines the OpenGL ES Common and Common-Lite profiles. A profile pipeline is described in the same order as in the OpenGL specification. The specification lists supported commands and state, and calls out commands and state that are part of the full (*desktop*) OpenGL specification but not part of the profile definition. This specification is *not* a standalone document describing the detailed behavior of the rendering pipeline subset and API. Instead, it provides a concise description of the differences between a full OpenGL renderer and the Common/Common-Lite renderer. This document is defined relative to the OpenGL 1.5 specification.

Starting with revision 1.1.10, a standalone document titled *OpenGL ES Common/Common-Lite Profile Specification (Full Specification)* has been derived from the OpenGL 1.5 specification. The *Full Specification* is the authoritative definition of OpenGL ES 1.1. This document, the *Difference Specification*, will continue to be maintained as a quick reference, and to enable direct comparisons with OpenGL 1.5.

This document specifies the OpenGL Common/Common-Lite renderer. A companion document defines one or more bindings to window system/OS platform combinations analogous to the GLX, WGL, and AGL specifications. [1] If required, an additional companion document describes utility library functionality analogous to the GLU specification.

1.1 Conventions

This document describes commands in the identical order as the OpenGL 1.5 specification. Each section corresponds to a section in the full OpenGL specification and describes the disposition of each command relative to Common/Common-Lite profile definition. Where necessary, the profile specification provides additional clarification of the reduced command behavior.

Each section of the specification includes tables summarizing the commands and parameters that are retained in the Common and Common-Lite profiles. Several symbols are used within the tables to indicate various special cases. The symbol † indicates that the floating-point form of the command is replaced by its fixed-point variant from the `OES_fixed_point` extension. The symbol $\diamondsuit$ indicates that the double-precision form of the command is replaced with its single-precision variant from the `OES_single_precision` extension. The symbol $*$ indicates that an enumerant is part of a new ES-specific extension. The superscript ‡ indicates that the command is supported subject to additional constraints described in the section body containing the table.

[1] See the *OpenGL ES Native Platform Graphics Interface specification*.

■ Additional material summarizing some of the reasoning behind certain decisions is included as an annotation at the end of each section, set in this typeface. ❑

Chapter 2

OpenGL Operation

The basic GL operation remains largely unchanged. Two significant changes in the Common and Common-Lite profiles are that commands cannot be accumulated in a display list for later processing, and the first stage of the pipeline for approximating curve and surface geometry is eliminated. The remaining pipeline stages include: per-vertex operations and primitive assembly, pixel operations, rasterization, per-fragment operations, and whole framebuffer operations.

The Common/Common-Lite profile introduces several OpenGL extensions that are defined relative to the full OpenGL 1.5 specification and then appropriately reduced to match the subset of commands in the profile. These OpenGL extensions are divided into two categories: those that are fully integrated into the profile definition – *core additions*; and those that remain extensions – *profile extensions*. Core additions do not use extension suffixes, whereas profile extensions retain their extension suffixes. Chapter 7 summarizes each extension and how it relates to the profile definition. Complete extension specifications are included in Appendix B.

■ The OpenGL ES profiles are part of a wider family of OpenGL-derived application programming interfaces. As such, the profiles share a similar processing pipeline, command structure, and the same OpenGL name space. Where necessary, extensions are created to augment the existing OpenGL 1.5 functionality. OpenGL ES-specific extensions play a role in OpenGL ES profiles similar to that played by OpenGL ARB extensions relative to the OpenGL specification. OpenGL ES-specific extensions are either precursors of functionality destined for inclusion in future core profile revisions, or formalization of important but non-mainstream functionality.

Extension specifications are written relative to the full OpenGL specification so that they can also be added as extensions to an OpenGL 1.5 implementation and so that they are easily adapted to profile functionality enhancements that are drawn from the full OpenGL specification. Extensions that are part of the core profile do not have extension suffixes, since they are not extensions to the profile, though they are extensions to OpenGL 1.5. ❑

2.1 OpenGL Fundamentals

Commands and tokens continue to be prefixed by **gl** and **GL_** in all profiles. The wide range of support for differing data types (8-bit, 16-bit, 32-bit and 64-bit; integer and floating-point) is reduced wherever possible to eliminate non-essential command variants and to reduce the complexity of the processing pipeline. Double-precision floating-point parameters and data types are eliminated completely, while other command and data type variations are considered on a command-by-command basis and eliminated when appropriate. In the Common-Lite variation of the Common profile, the floating-point data type is also eliminated in favor

of the fixed-point data type described in the `OES_fixed_point` extension specification.

2.1.1 Fixed-Point Computation

Both the Common and Common-Lite profile support fixed-point vertex attributes and command parameters using a 32-bit two's-complement signed representation with 16 bits to the right of the binary point (fraction bits). The Common profile pipeline retains the same range and precision requirements as specified in Section 2.1.1 of the OpenGL 1.5 specification. The Common-Lite profile pipeline must meet the range and precision requirements specified in the `OES_fixed_point` extension:

> Internal computations can use either fixed-point or floating-point arithmetic. Fixed-point computations must be accurate to within $\pm 2^{-15}$. The maximum representable magnitude for a fixed-point number used to represent positional or normal coordinates must be at least 2^{15}; the maximum representable magnitude for colors or texture coordinates must be at least 2^{10}. The maximum representable magnitude for all other fixed-point values must be at least 2^{15}. $x \cdot 0 = 0 \cdot x = 0$. $1 \cdot x = x \cdot 1 = x$. $x + 0 = 0 + x = x$. $0^0 = 1$. Fixed-point computations may lead to overflows or underflows. The results of such computations are undefined, but must not lead to GL interruption or termination.

Furthermore, the following additional constraint must be met for both profiles:

> Using the notation 16.16 to indicate a 32-bit two's-complement fixed-point number with 16 bits of fraction; if an incoming vertex is representable using 16.16, the modelview and projection matrices are representable in 16.16, and the resulting eye-space and NDC-space vertices are representable in 16.16 (when computed using intermediate representations with sufficient dynamic range), then the transformation pipeline must compute the eye-space and NDC-space vertices to some reasonable accuracy (i.e., overflow is not acceptable).

■ The Common-Lite profile is a fixed-point profile. The precision and dynamic range requirements are minimal to allow a broad range of implementations, while strong enough to allow portable application behavior for applications written strictly to the minimum behavior. The accuracy requirements allow pipeline implementations to internally use either fixed-point or floating-point arithmetic. The Common profile is a superset of the Common-Lite profile and requires floating-point-like dynamic range to avoid unexpected behavior in applications using floating-point input. ❑

2.2 GL State

The Common and Common-Lite profiles retain a large subset of the client and server state described in the full OpenGL specification. The separation of client and server state persists. Section 6.2 summarizes the disposition of all state variables relative to the Common/Common-Lite profile.

2.3 GL Command Syntax

The OpenGL command and type naming conventions are retained identically. The new types `fixed` and `clampx` are added with the corresponding command suffix, 'x'. Commands using the suffixes for the types: `byte`, `ubyte`, `short`, and `ushort` are not supported, except for glColor4ub. The type `double` and all double-precision commands are eliminated. The result is that the Common profile uses only the suffixes 'f', 'i', and 'x' and the Common-Lite profile uses only the suffixes 'i' and 'x'.

2.4 Basic GL Operation

The basic command operation remains identical to OpenGL 1.5. The major differences from the OpenGL 1.5 pipeline are that commands cannot be placed in a display list; there is no polynomial function evaluation stage; and blocks of fragments cannot be sent directly to the individual fragment operations.

2.5 GL Errors

The full OpenGL error detection behavior is retained, including ignoring offending commands and setting the current error state. In all commands, parameter values that are not supported by the profile are treated like any other unrecognized parameter value and an error results, i.e., `INVALID_ENUM` or `INVALID_VALUE`. Table 2.1 lists the errors.

OpenGL 1.5	Common	Common-Lite
`NO_ERROR`	✓	✓
`INVALID_ENUM`	✓	✓
`INVALID_VALUE`	✓	✓
`INVALID_OPERATION`	✓	✓
`STACK_OVERFLOW`	✓	✓
`STACK_UNDERFLOW`	✓	✓
`OUT_OF_MEMORY`	✓	✓
`TABLE_TOO_LARGE`	–	–

Table 2.1: Error Disposition

The command **GetError** is retained to return the current error state. As in OpenGL 1.5, it may be necessary to call **GetError** multiple times to retrieve error state from all parts of the pipeline.

OpenGL 1.5	Common	Common-Lite
GetError `(void)`	✓	✓

■ Well-defined error behavior allows portable applications to be written. Retrievable error state allows application developers to debug commands with invalid parameters during development. This is an important feature during initial profile deployment. ❑

2.6 Begin/End Paradigm

The Common and Common-Lite profiles draw geometric objects exclusively using vertex arrays. Associated colors, normals, and texture coordinates are specified using vertex arrays. The associated auxiliary values for color, normal, and texture coordinate can also be set using a small subset of the associated attribute specification commands described in Section 2.7. Since the commands **Begin** and **End** are not supported, no internal state indicating the begin/end state is maintained.

The `POINTS`, `LINES`, `LINE_STRIP`, `LINE_LOOP`, `TRIANGLES`, `TRIANGLE_STRIP`, and `TRIANGLE_FAN` primitives are supported. The `QUADS`, `QUAD_STRIP`, and `POLYGON` primitives are not supported.

Color index rendering is not supported. Edge flags are not supported.

OpenGL 1.5	Common	Common-Lite
Begin(`enum mode`)	–	–
End(`void`)	–	–
EdgeFlag[v](`T flag`)	–	–

■ The Begin/End paradigm, while convenient, leads to a large number of commands that need to be implemented. Correct implementation also involves suppression of commands that are not legal between Begin and End. Tracking this state creates an additional burden on the implementation. Vertex arrays, arguably can be implemented more efficiently since they present all of the primitive data in a single function call. Edge flags are not included, as they are only used when drawing polygons as outlines and support for **PolygonMode** has not been included.

Quads and polygons are eliminated since they can be readily emulated with triangles and it reduces an ambiguity with respect to decomposition of these primitives to triangles, since it is entirely left to the application. Elimination of quads and polygons removes special cases for line mode drawing requiring edge flags (should **PolygonMode** be re-instated). ❑

2.7 Vertex Specification

The Common profile does not include the concept of Begin and End. Vertices are specified using vertex arrays exclusively. Only `float`, `short`, and `byte` coordinate and component types are supported with the exception of `ubyte` rather than `short` color components. There is limited support for specifying the current color, normal, and texture coordinate using the fixed-point or floating-point forms of the commands **Color4**, **Normal3**, and **MultiTexCoord4**.

OpenGL 1.5	Common	Common-Lite
Vertex{**234**}{**sifd**}**[v]**(`T coords`)	–	–
Normal3f(`float coords`)	✓	†
Normal3{**bsifd**}**[v]**(`T coords`)	–	–
TexCoord{**1234**}{**sifd**}**[v]**(`T coords`)	–	–
MultiTexCoord4f(`enum texture, float coords`)	✓	†
MultiTexCoord123{**sifd**}**[v]**(`enum texture, T coords`)	–	–
MultiTexCoord4{**sid**}**[v]**(`enum texture, T coords`)	–	–
Color4f(`float components`)	✓	†
Color4ub(`T components`)	✓	✓
Color{**34**}{**bsifd ub us ui**}**[v]**(`T components`)	–	–
FogCoord{**fd**}**[v]**(`T coord`)	–	–
SecondaryColor3{**bsifd ub us ui**}**[v]**(`T components`)	–	–
Index{**sifd ub**}**[v]**(`T components`)	–	–

■ A handful of *fine grain* commands (**Color**, **Normal**, **MultiTexCoord**) are included so that per-primitive attributes can be set. For each command, the most general form of the floating-point version of the command is retained to simplify addition of extensions or future revisions. Since these commands are unlikely to be issued frequently, as they can only be used to set (overall) per-primitive attributes, performance is not an issue.

The Common and Common-Lite profiles support only the RGBA rendering model. One or more of the RGBA component depths may be zero. Color index rendering is not supported. ❑

2.8 Vertex Arrays

The `OES_byte_coordinates` extension allows vertex, normal and texture coordinates to be specified using `byte` types. Color index and edge flags are not supported. Both indexed and non-indexed arrays are supported, but the **InterleavedArrays** and **ArrayElement** commands are not supported.

OpenGL 1.5	Common	Common-Lite
VertexPointer `(int size, enum type, sizei stride, const void *ptr)`		
`size = 2,3,4 type = BYTE`	∗	∗
`size = 2,3,4 type = SHORT`	✓	✓
`size = 2,3,4 type = FLOAT`	✓	†
`size = * type = INT,DOUBLE`	–	–
NormalPointer `(enum type, sizei stride, const void *ptr)`		
`type = SHORT,BYTE`	✓	✓
`type = FLOAT`	✓	†
`type = INT,DOUBLE`	–	–
ColorPointer `(int size, enum type, sizei stride, const void *ptr)`		
`size = 4 type = UNSIGNED_BYTE`	✓	✓
`size = 4 type = FLOAT`	✓	†
`size = 3 type = FLOAT,UNSIGNED_BYTE`	–	–
`type = INT, DOUBLE`	–	–
TexCoordPointer `(int size, enum type, sizei stride, const void *ptr)`		
`size = 2,3,4 type = BYTE`	∗	∗
`size = 2,3,4 type = SHORT`	✓	✓
`size = 2,3,4 type = FLOAT`	✓	†
`size = 1 type = *`	–	–
SecondaryColorPointer `(int size, enum type, sizei stride, void *ptr)`	–	–
FogCoordPointer `(enum type, sizei stride, void *ptr)`	–	–
EdgeFlagPointer `(sizei stride, const void *ptr)`	–	–
IndexPointer `(enum type, sizei stride, const void *ptr)`	–	–
ArrayElement `(int i)`	–	–
DrawArrays `(enum mode, int first, sizei count)`		
`mode = POINTS,LINES,LINE_STRIP,LINE_LOOP`	✓	✓
`mode = TRIANGLES,TRIANGLE_STRIP,TRIANGLE_FAN`	✓	✓
`mode = QUADS,QUAD_STRIP,POLYGON`	–	–
DrawElements `(enum mode, sizei count, enum type, const void *indices)`		
`mode = POINTS,LINES,LINE_STRIP,LINE_LOOP`	✓	✓
`mode = TRIANGLES,TRIANGLE_STRIP,TRIANGLE_FAN`	✓	✓
`mode = QUADS,QUAD_STRIP,POLYGON`	–	–
`type = UNSIGNED_BYTE,UNSIGNED_SHORT`	✓	✓
`type = UNSIGNED_INT`	–	–
MultiDrawArrays `(enum mode, int *first, sizei *count, sizei primcount)`	–	–
MultiDrawElements `(enum mode, sizei *count, enum type, void **indices, sizei primcount)`	–	–

OpenGL 1.5	Common	Common-Lite
InterleavedArrays(enum format, sizei stride, const void *pointer)	–	–
DrawRangeElements(enum mode, uint start, uint end, sizei count, enum type, const void *indices)	–	–
ClientActiveTexture(enum texture)	✓	✓
EnableClientState(enum cap)		
cap = TEXTURE_COORD_ARRAY,COLOR_ARRAY	✓	✓
cap = NORMAL_ARRAY,VERTEX_ARRAY	✓	✓
cap = EDGE_FLAG_ARRAY,INDEX_ARRAY	–	–
cap = FOG_COORD_ARRAY,SECONDARY_COLOR_ARRAY	–	–
DisableClientState(enum cap)		
cap = TEXTURE_COORD_ARRAY,COLOR_ARRAY	✓	✓
cap = NORMAL_ARRAY,VERTEX_ARRAY	✓	✓
cap = EDGE_FLAG_ARRAY, INDEX_ARRAY	–	–
cap = FOG_COORD_ARRAY,SECONDARY_COLOR_ARRAY	–	–

■ `Float` types are supported for all-around generality, `short` and `byte` types are supported for space efficiency. Four-component vertex and texture coordinates are supported to allow an application to fully specify post-projection vertex and texture coordinates before division by w or q. Support for indexed vertex arrays allows for greater reuse of coordinate data between multiple faces, that is, when the shared edges are smooth. The indexing support is limited to `ubyte` and `ushort` indices since there is typically enough locality in the vertex array data to address the vertices with these more compact index types.

The OpenGL 1.5 specification defines the initial type for each of the arrays to be `FLOAT` for the common profile and `FIXED` for the common-lite profile.

Multitexture with a minimum of two texture units is required by OpenGL ES 1.1. ❑

2.9 Buffer Objects

The vertex data arrays described in Section 2.8 are stored in client memory. It is sometimes desirable to store frequently used client data, such as vertex array data in high-performance server memory. GL buffer objects provide a mechanism that clients can use to allocate, initialize and render from memory. Buffer objects can be used to store vertex array and element index data.

OpenGL 1.5	Common	Common-Lite
BindBuffer(enum target, uint buffer)	✓	✓
DeleteBuffers(sizei n, const uint *buffers)	✓	✓
GenBuffers(sizei n, uint *buffers)	✓	✓
BufferData(enum target, sizeiptr size, const void *data, enum usage)	✓	✓
BufferSubData(enum target, intptr offset, sizeiptr size, const void *data)	✓	✓
MapBuffer(enum target, enum access)	–	–
UnmapBuffer(enum target)	–	–

Name	Type	Initial Value	Legal Values
BUFFER_SIZE	integer	0	any non-negative integer
BUFFER_USAGE	enum	STATIC_DRAW	STATIC_DRAW, DYNAMIC_DRAW

Table 2.2: Buffer object parameters and their values

■ The STREAM_DRAW, STREAM_COPY, STREAM_READ, STATIC_COPY, STATIC_READ, DYNAMIC_COPY, and DYNAMIC_READ tokens and the **MapBuffer** and **UnmapBuffer** functions are not supported because it may not be possible for an application to read or get a pointer to the vertex data from the vertex buffers in server memory.

BufferData and BufferSubData define two new types that will work well on 64-bit systems, analogous to C's "intptr_t". The new type "GLintptrARB" should be used in place of GLint whenever it is expected that values might exceed 2 billion. The new type "GLsizeiptrARB" should be used in place of GLsizei whenever it is expected that counts might exceed 2 billion. Both types are defined as signed integers large enough to contain any pointer value. As a result, they naturally scale to larger numbers of bits on systems with 64-bit or even larger pointers. ❑

2.10 Rectangles

The commands for directly specifying rectangles are not supported.

OpenGL 1.5	Common	Common-Lite
Rect{**sifd**}(T x1, T y1, T x2, T y2)	–	–
Rect{**sifd**}**v**(T v1[2], T v2[2])	–	–

■ The rectangle commands are not used enough in applications to justify maintaining a redundant mechanism for drawing a rectangle. ❑

2.11 Coordinate Transformations

The full transformation pipeline is supported with the following exceptions: no support for specification of double-precision matrices and transformation parameters; no support for the transpose form of the **LoadMatrix** and **MultMatrix** commands; no support for COLOR matrix; and no support for texture coordinate generation. The double-precision only commands **DepthRange**, **Frustum**, and **Ortho** are replaced with single-precision or fixed-point variants from the OES_single_precision and OES_fixed_point extensions. The minimum depth of the MODELVIEW matrix stack is changed from 32 to 16.

OpenGL 1.5	Common	Common-Lite
DepthRange(clampd n, clampd f)	♢	†
Viewport(int x, int y, sizei w, sizei h)	✓	✓
MatrixMode(enum mode)		
mode = MODELVIEW,PROJECTION,TEXTURE	✓	✓
mode = COLOR	–	–
LoadMatrixf(float m[16])	✓	†

OpenGL 1.5	Common	Common-Lite
LoadMatrixd (double m[16])	–	–
MultMatrixf (float m[16])	✓	†
MultMatrixd (double m[16])	–	–
LoadTransposeMatrix{fd} (T m[16])	–	–
MultTransposeMatrix{fd} (T m[16])	–	–
LoadIdentity (void)	✓	✓
Rotatef (float angle, float x, float y, float z)	✓	†
Rotated (double angle, double x, double y, double z)	–	–
Scalef (float x, float y, float z)	✓	†
Scaled (double x, double y, double z)	–	–
Translatef (float x, float y, float z)	✓	†
Translated (double x, double y, double z)	–	–
Frustum (double l, double r, double b, double t, double n, double f)	♢	†
Ortho (double l, double r, double b, double t, double n, double f)	♢	†
ActiveTexture (enum texture)	✓	✓
PushMatrix (void)		
TEXTURE and PROJECTION (2 deep)	✓	✓
MODELVIEW (16 deep)	✓	✓
PopMatrix (void)	✓	✓
Enable/Disable (RESCALE_NORMAL)	✓	✓
Enable/Disable (NORMALIZE)	✓	✓
TexGen{ifd}[v] (enum coord, enum pname, T param)	–	–
GetTexGen{ifd}v (enum coord, enum pname, T *params)	–	–
Enable/Disable (TEXTURE_GEN_{STRQ})	–	–

■ The double-precision version of the transform commands are not necessary when there is a single precision version. The matrix stacks and convenience functions for computing rotations, scales, and translations, as well as projection matrices are kept in their entirety since they are used by a large number of of applications. The minimum depth for the modelview stack is reduced from 32 to 16 to reduce the storage requirements somewhat. The projection and texture stack depths are already limited to a depth of two. The non-transpose form of the matrix load and multiply commands are retained over the transpose versions to maximize compatibility with existing programming practices.

The viewport and depth range commands are supported since they provide necessary application control over where primitives are drawn. While texture coordinate generation is useful, it is considered too much of an implementation burden (applications can implement it to some extent themselves). Texgen is a strong candidate for the next revision. Both normalization and rescaling of normals are supported since normalization is deemed necessary and rescaling can be implemented using normalization minimizing implementation burden. ❑

2.12 Clipping

Primitives are clipped to the *clip volume*. In clip coordinates, the *view volume* is defined by

$$-w_c <= x_c <= w_c$$

$$-w_c <= y_c <= w_c$$

$$-w_c <= z_c <= w_c$$

This view volume may be further restricted by as many as *n* client-defined clip planes to generate the clip volume. *n* is an implementation defined maximum that must be at least 1.

OpenGL 1.5	Common	Common-Lite
ClipPlane(enum plane, const double *equation)	♢	†
GetClipPlane(enum plane, double *equation)	♢	†
Enable/Disable(CLIP_PLANE{0...n-1})	✓	✓

■ User-specified clipping planes are used predominantly in engineering and scientific applications. However, a single clipping plane is useful for some ”portal-culling” algorithms. ❑

2.13 Current Raster Position

The concept of the current raster position for positioning pixel rectangles and bitmaps is not supported. Current raster state and commands for setting the raster position are not supported.

OpenGL 1.5	Common	Common-Lite
RasterPos{**2,3,4**}{**sifd**}[**v**](T coords)	–	–
WindowPos{**2,3**}{**sifd**}[**v**](T coords)	–	–

■ Bitmaps and pixel image primitives are not supported so there is no need to specify the raster position. ❑

2.14 Colors and Coloring

The OpenGL 1.5 lighting model is supported with the following exceptions: no support for the color index lighting, secondary color, different front and back materials, local viewer, or color material mode other than AMBIENT_AND_DIFFUSE.

Directional, positional, and spot lights are all supported. An implementation must support a minimum of 8 lights. The **Material** command cannot independently change the front and back face properties, so the result is that materials always have the same front and back properties. Two-sided lighting is supported, though the front and back material properties used in the lighting computation will also be equal. The **ColorMaterial** command is not supported, so the color material mode cannot be changed from the default AMBIENT_AND_DIFFUSE mode, though COLOR_MATERIAL can be enabled in this mode. Neither local viewing computations nor separate specular color computation can be enabled using the **LightModel** command, therefore only the OpenGL 1.5 default infinite viewer and single color computational models are supported. Smooth and flat shading are fully supported for all primitives.

OpenGL 1.5	Common	Common-Lite
FrontFace `(enum mode)`	✓	✓
Enable/Disable `(LIGHTING)`	✓	✓
Enable/Disable `(LIGHT{0-7})`	✓	✓
Materialf[v] `(enum face, enum pname, T param)`		
`face = FRONT_AND_BACK`	✓	†
`face = FRONT,BACK`	–	–
`pname = AMBIENT,DIFFUSE,SPECULAR,EMISSION,SHININESS`	✓	†
`pname = AMBIENT_AND_DIFFUSE`	✓	†
`pname = COLOR_INDEXES`	–	–
Materiali[v] `(enum face, enum pname, T param)`	–	–
GetMaterialfv `(enum face, enum pname, T *params)`	✓	†
GetMaterialiv `(enum face, enum pname, T *params)`	–	–
Lightf[v] `(enum light, enum pname, T param)`	✓	†
Lighti[v] `(enum light, enum pname, T param)`	–	–
GetLightfv `(enum light, enum pname, T *params)`	✓	†
GetLightiv `(enum light, enum pname, T *params)`	–	–
LightModelf[v] `(enum pname, T param)`		
`pname = LIGHT_MODEL_TWO_SIDE`	✓	†
`pname = LIGHT_MODEL_AMBIENT`	✓	†
`pname = LIGHT_MODEL_COLOR_CONTROL`	–	–
`pname = LIGHT_MODEL_LOCAL_VIEWER`	–	–
LightModeli[v] `(enum pname, T param)`	–	–
Enable/Disable `(COLOR_MATERIAL)`	✓‡	✓‡
ColorMaterial `(enum face, enum mode)`	–	–
ShadeModel `(enum mode)`	✓	✓

■ Lighting is a desirable feature, so as much as possible is included in the Common and Common-Lite profiles. The minimum number of lights is not reduced since reducing it only saves memory for the state and the savings is not significant unless it is greatly reduced. The number cannot be greatly reduced (e.g., to 1 or 2) as many applications need three or more lights. Support for secondary color creates a non-trivial burden in the rasterization stage of the pipeline so it is not included. Local viewer increases the amount of computation in the lighting pipeline and is not widely used (usually because of the performance degradation), the other features controlled by the **LightModel** (scene ambient and two-sided lighting) are retained. Scene ambient is retained since its default value is non-zero and there would be no method to disable its effect if it were not included. Two-sided lighting is retained in a simplified fashion – the front and back material values must always be equal. To ensure this, only `FRONT_AND_BACK` can be used as the face parameter.

The most common use for the **ColorMaterial** functionality is to change the ambient and diffuse coefficients of the material. Since this is the default mode of the command, the **ColorMaterial** command is not included, but the ability to enable and disable it is, so the net effect is that only the ambient and diffuse material parameters can be modified. ❑

Chapter 3

Rasterization

3.1 Invariance

The invariance rules are retained in full.

3.2 Antialiasing

Multisampling is supported though an implementation is not required to provide a multisample buffer.

OpenGL 1.5	Common	Common-Lite
Enable/Disable (`MULTISAMPLE`)	✓	✓

■ Multisampling is a desirable feature. Since an implementation need not provide an actual multisample buffer and the command overhead is low, it is included. ❑

3.3 Points

Aliased and antialiased points are fully supported. The requested point size can also be multiplied with a distance attenuation factor, clamped to a specified point size range, and further clamped to the implementation dependent point size range to produce the derived point size. Details of how to do distance attenuation of point size is described in section 3.3 of the OpenGL 1.5 specification.

OpenGL 1.5	Common	Common-Lite
PointSize (`float size`)	✓	†
PointParameterf[v] (`enum pname, T param`)	✓	†
PointParameteri[v] (`enum pname, T param`)	–	–
Enable/Disable (`POINT_SMOOTH`)	✓	✓

■ See below. ❑

OpenGL 1.5	Common	Common-Lite
LineWidth (`float width`)	✓	†
Enable/Disable (`LINE_SMOOTH`)	✓	✓
LineStipple (`int factor, ushort pattern`)	–	–
Enable/Disable (`LINE_STIPPLE`)	–	–

3.4 Line Segments

Aliased and antialiased lines are fully supported. Line stippling is not supported.

■ Antialiasing is important for visual quality, particularly for devices with low spatial resolution (pixels per mm). Some antialiasing can be implemented within the application using 2D textures, but antialiasing is used by enough applications that it should be in the profile rather than something left to the application. The OpenGL 1.5 point and line antialiasing requirements provide substantial implementation latitude. In particular, only size/width 1.0 is required to be supported and the coverage computation constraints are easily satisfied. Line stippling is also used by ”presentation graphics” and engineering applications. It can be implemented by the application, and the implementation cost is considered too high to include in the profile. ❑

3.5 Polygons

Polygonal geometry support is reduced to triangle strips, triangle fans and independent triangles. All rasterization modes are supported except for point and line **PolygonMode** and antialiased polygons using polygon smooth. Depth offset is supported in `FILL` mode only.

OpenGL 1.5	Common	Common-Lite
CullFace (`enum mode`)	✓	✓
Enable/Disable (`CULL_FACE`)	✓	✓
PolygonMode (`enum face, enum mode`)	–	–
Enable/Disable (`POLYGON_SMOOTH`)	–	–
PolygonStipple (`const ubyte *mask`)	–	–
GetPolygonStipple (`ubyte *mask`)	–	–
Enable/Disable (`POLYGON_STIPPLE`)	–	–
PolygonOffset (`float factor, float units`)	✓	†
Enable/Disable (`enum cap`)		
`cap = POLYGON_OFFSET_FILL`	✓	✓
`cap = POLYGON_OFFSET_LINE, POLYGON_OFFSET_POINT`	–	–

■ Support for all triangle types (independents, strips, fans) is not overly burdensome and each type has some desirable utility: strips for general performance and applicability, independents for efficiently specifying unshared vertex attributes, and fans for representing ”corner-turning” geometry. Face culling is important for eliminating unnecessary rasterization. Polygon stipple is desirable for doing patterned fills for ”presentation graphics”. It is also useful for transparency, but support for alpha is sufficient for that. Polygon stippling does represent a large burden for the polygon rasterization path and can usually be emulated using texture mapping and alpha test, so it is omitted. Polygon offset for filled triangles is necessary for rendering coplanar and outline polygons and if not present requires either stencil buffers or application tricks. Antialiased polygons using `POLYGON_SMOOTH` is just as

desirable as antialiasing for other primitives, but is too large an implementation burden to include in the Common/Common-Lite profile. ❑

3.6 Pixel Rectangles

No support for directly drawing pixel rectangles is included. Limited **PixelStore** support is retained to allow different pack alignments for **ReadPixels** and unpack alignments for **TexImage2D**. **DrawPixels**, **PixelTransfer** modes and **PixelZoom** are not supported. The Imaging subset is not supported.

OpenGL 1.5	Common	Common-Lite
PixelStorei(enum pname, T param)		
pname = PACK_ALIGNMENT,UNPACK_ALIGNMENT	✓	✓
pname = <all other values>	–	–
PixelStoref(enum pname, T param)	–	–
PixelTransfer{**if**}(enum pname, T param)	–	–
PixelMap{**ui us f**}**v**(enum map, int size, T *values)	–	–
GetPixelMap{**ui us f**}**v**(enum map, T *values)	–	–
Enable/Disable(COLOR_TABLE)	–	–
ColorTable(enum target, enum internalformat, sizei width, enum format, enum type, const void *table)	–	–
ColorSubTable(enum target, sizei start, sizei count, enum format, enum type, const void *data)	–	–
ColorTableParameter{**if**}**v**(enum target, enum pname, T *params)	–	–
GetColorTableParameter{**if**}**v**(enum target, enum pname, T *params)	–	–
CopyColorTable(enum target, enum internalformat, int x, int y, sizei width)	–	–
CopyColorSubTable(enum target, sizei start, int x, int y, sizei width)	–	–
GetColorTable(enum target, enum format, enum type, void *table)	–	–
ConvolutionFilter1D(enum target, enum internalformat, sizei width, enum format, enum type, const void *image)	–	–
ConvolutionFilter2D(enum target, enum internalformat, sizei width, sizei height, enum format, enum type, const void *image)	–	–
GetConvolutionFilter(enum target, enum format, enum type, void*image)	–	–
CopyConvolutionFilter1D(enum target, enum internalformat, int x, int y, sizei width)	–	–

OpenGL 1.5	Common	Common-Lite
CopyConvolutionFilter2D(enum target, enum internalformat, int x, int y, sizei width, sizei height)	–	–
SeparableFilter2D(enum target, enum internalformat, sizei width, sizei height, enum format, enum type, const void *row, const void *column)	–	–
GetSeparableFilter(enum target, enum format, enum type, void *row, void *column, void *span)	–	–
ConvolutionParameter{**if**}**[v]**(enum target, enum pname, T param)	–	–
GetConvolutionParameter{**if**}**v**(enum target, enum pname, T *params)	–	–
Enable/Disable(POST_CONVOLUTION_COLOR_TABLE)	–	–
MatrixMode(COLOR)	–	–
Enable/Disable(POST_COLOR_MATRIX_COLOR_TABLE)	–	–
Enable/Disable(HISTOGRAM)	–	–
Histogram(enum target, sizei width, enum internalformat, boolean sink)	–	–
ResetHistogram(enum target)	–	–
GetHistogram(enum target, boolean reset, enum format, enum type, void *values)	–	–
GetHistogramParameter{**if**}**v**(enum target, enum pname, T *params)	–	–
Enable/Disable(MINMAX)	–	–
Minmax(enum target, enum internalformat, boolean sink)	–	–
ResetMinmax(enum target)	–	–
GetMinmax(enum target, boolean reset, enum format, enum types, void *values)	–	–
GetMinmaxParameter{**if**}**v**(enum target, enum pname, T *params)	–	–
DrawPixels(sizei width, sizei height, enum format, enum type, void *data)	–	–
PixelZoom(float xfactor, float yfactor)	–	–

■ The OpenGL 1.5 specification includes substantial support for operating on pixel images. In the Common and Common-Lite profiles, the ability to draw pixel images is important, but with the constraint of minimizing the implementation burden. There is a concern that **DrawPixels** is often poorly implemented on hardware accelerators and that many applications are better served by emulating **DrawPixels** functionality by initializing a texture image with the host image and then drawing the texture image to a screen-aligned quadrilateral. This has the advantage of eliminating the **DrawPixels** processing path and and allows the image to be cached and drawn multiple times without

re-transferring the image data from the application's address space. However, it requires extra processing by the application and the implementation, possibly requiring the image to be copied twice. The OES_draw_texture extension, added to OpenGL ES 1.1, addresses the above issues and provides an efficient mechanism to draw pixel images as textures.

The command **PixelStore** must be included to allow changing the pack alignment for **ReadPixels** and unpack alignment for **TexImage2D** to something other than the default value of 4 to support `ubyte RGB` image formats. The integer version of **PixelStore** is retained rather than the floating-point version since all parameters can be fully expressed using integer values. ❑

3.7 Bitmaps

Bitmap images are not supported.

OpenGL 1.5	Common	Common-Lite
Bitmap(`sizei width, sizei height, float xorig, float yorig, float xmove, float ymove, const ubyte *bitmap`)	–	–

■ The **Bitmap** command is useful for representing image data compactly and for positioning images directly in window coordinates. Since **DrawPixels** is not supported, the positioning functionality is not required. A strong enough case hasn't been made for the ability to represent font glyphs or other data more efficiently before transfer to the rendering pipeline. ❑

3.8 Texturing

OpenGL ES 1.1 requires a minimum of two texture units to be supported. 1D textures, 3D textures, and cube maps are not supported. 2D textures are supported with the following exceptions: only a limited number of image format and type combinations are supported, listed in Table 3.1. Table 3.2 summarizes the disposition of all image types. The only internal formats supported are the base internal formats: `RGBA`, `RGB`, `LUMINANCE`, `ALPHA`, and `LUMINANCE_ALPHA`. The format must match the base internal format (no conversions from one format to another during texture image processing are supported) as described in Table 3.1. If the texture format does not match the base internal format an `INVALID_OPERATION` error results. Texture borders are not supported (the **border** parameter must be zero, and an `INVALID_VALUE` error results if it is non-zero).

3.8.1 Copy Texture

CopyTexture and **CopyTexSubImage** are supported. The internal format parameter can be any of the base internal formats described for **TexImage2D** subject to the constraint that color buffer components can be dropped during the conversion to the base internal format, but new components cannot be added. For example, an RGB color buffer can be used to create `LUMINANCE` or `RGB` textures, but not `ALPHA`, `LUMINANCE_ALPHA`, or `RGBA` textures. Table 3.3 summarizes the allowable framebuffer and base internal format combinations. If the framebuffer format is not compatible with the base texture format an `INVALID_OPERATION` error results.

OpenGL 1.5	Common	Common-Lite
`UNSIGNED_BYTE`	✓	✓

OpenGL 1.5	Common	Common-Lite
BITMAP	–	–
BYTE	–	–
UNSIGNED_SHORT	–	–
SHORT	–	–
UNSIGNED_INT	–	–
INT	–	–
FLOAT	–	–
UNSIGNED_BYTE_3_3_2	–	–
UNSIGNED_BYTE_3_3_2_REV	–	–
UNSIGNED_SHORT_5_6_5	✓	✓
UNSIGNED_SHORT_5_6_5_REV	–	–
UNSIGNED_SHORT_4_4_4_4	✓	✓
UNSIGNED_SHORT_4_4_4_4_REV	–	–
UNSIGNED_SHORT_5_5_5_1	✓	✓
UNSIGNED_SHORT_5_5_5_1_REV	–	–
UNSIGNED_INT_8_8_8_8	–	–
UNSIGNED_INT_8_8_8_8_REV	–	–
UNSIGNED_INT_10_10_10_2	–	–
UNSIGNED_INT_10_10_10_2_REV	–	–

Table 3.2: Image Types

3.8.2 Compressed Textures

Compressed textures are supported including sub-image specification; however, no method for reading back a compressed texture image is included, so implementation vendors must provide separate tools for creating compressed images. The generic compressed internal formats are not supported, so compression of textures using **TexImage2D** is not supported. The OES_compressed_paletted_texture extension defines several compressed texture formats.

3.8.3 Texture Addressing Modes

Wrap modes REPEAT and CLAMP_TO_EDGE are supported, but not CLAMP, CLAMP_TO_BORDER and MIRRORED_REPEAT. Wrap mode for "R" coordinate i.e. TEXTURE_WRAP_R is not supported. Texture priorities, LOD clamps, and explicit base and maximum level specification are not supported. The remaining OpenGL 1.5 texture parameters are supported including all filtering modes. Texture objects are supported, but proxy textures are not supported.

3.8.4 Texture Completeness

For 2D textures, a texture is *complete* in OpenGL ES if the following conditions all hold true:

- the set of mipmap arrays are specified with the same type.
- the dimensions of the arrays follow the sequence described in the **Mimapping** discussion of section 3.8.7 of the OpenGL 1.5 specification.

Internal Format	External Format	Type	Bytes per Pixel
RGBA	RGBA	UNSIGNED_BYTE	4
RGB	RGB	UNSIGNED_BYTE	3
RGBA	RGBA	UNSIGNED_SHORT_4_4_4_4	2
RGBA	RGBA	UNSIGNED_SHORT_5_5_5_1	2
RGB	RGB	UNSIGNED_SHORT_5_6_5	2
LUMINANCE_ALPHA	LUMINANCE_ALPHA	UNSIGNED_BYTE	2
LUMINANCE	LUMINANCE	UNSIGNED_BYTE	1
ALPHA	ALPHA	UNSIGNED_BYTE	1

Table 3.1: Texture Image Formats and Types

	Texture Format				
Color Buffer	**A**	**L**	**LA**	**RGB**	**RGBA**
A	✓	–	–	–	–
L	–	✓	–	–	–
LA	✓	✓	✓	–	–
RGB	–	✓	–	✓	–
RGBA	✓	✓	✓	✓	✓

Table 3.3: CopyTexture Internal Format/Color Buffer Combinations

The check for completeness is done when a given texture is used to render geometry.

3.8.5 Texture State

The state necessary for texture can be divided into two categories. First, there is the set of mipmap arrays (for the two-dimensional texture target) and their number. Each array has associated with it a width, height, an integer describing the internal format of the image, a boolean describing whether the image is compressed or not, and an integer size of a compressed image. Each initial texture array is null (zero width, and height, internal format is undefined, with the compressed flag set to FALSE, a zero compressed size, and zero-sized components). Next, we have the texture properties which consists of the selected minification and magnification filters, the wrap modes for s, and t, a boolean flag indicating whether the texture is resident, and a boolean indicating whether automatic mipmap generation should be performed. The value of the resident flag is determined by the GL and may change as a result of other GL operations. The flag may only be queried, not set, by applications. In the initial state, the value assigned to TEXTURE_MIN_FILTER is NEAREST_MIPMAP_LINEAR, and the value for TEXTURE_MAG_FILTER is LINEAR. s, and t wrap modes are all set to REPEAT.

OpenGL 1.5	Common	Common-Lite
TexImage1D(enum target, int level, int internalFormat, sizei width, int border, enum format, enum type, const void *pixels)	–	–
TexImage2D(enum target, int level, int internalFormat, sizei width, sizei height, int border, enum format, enum type, const void *pixels)		

OpenGL 1.5	Common	Common-Lite
`target = TEXTURE_2D border = 0`	✓‡	✓‡
`target = PROXY_TEXTURE_2D`	–	–
`border > 0`	–	–
TexImage3D`(enum target, int level, enum internalFormat, sizei width, sizei height, sizei depth, int border, enum format, enum type, const void *pixels)`	–	–
GetTexImage`(enum target, int level, enum format, enum type, void *pixels)`	–	–
TexSubImage1D`(enum target, int level, int xoffset, sizei width, enum format, enum type, const void *pixels)`	–	–
TexSubImage2D`(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, enum type, const void *pixels)`	✓‡	✓‡
TexSubImage3D`(enum target, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, enum type, const void *pixels)`	–	–
CopyTexImage1D`(enum target, int level, enum internalformat, int x, int y, sizei width, int border)`	–	–
CopyTexImage2D`(enum target, int level, enum internalformat, int x, int y, sizei width, sizei height, int border)`		
`border = 0`	✓‡	✓‡
`border > 0`	–	–
CopyTexSubImage1D`(enum target, int level, int xoffset, int x, int y, sizei width)`	–	–
CopyTexSubImage2D`(enum target, int level, int xoffset, int yoffset, int x, int y, sizei width, sizei height)`	✓‡	✓‡
CopyTexSubImage3D`(enum target, int level, int xoffset, int yoffset, int zoffset, int x, int y, sizei width, sizei height)`	–	–
CompressedTexImage1D`(enum target, int level, enum internalformat, sizei width, int border, sizei imageSize, const void *data)`	–	–
CompressedTexImage2D`(enum target, int level, enum internalformat, sizei width, sizei height, int border, sizei imageSize, const void *data)`		
`target = TEXTURE_2D border = 0`	✓‡	✓‡
`target = PROXY_TEXTURE_2D`	–	–
`border > 0`	–	–

OpenGL 1.5	Common	Common-Lite
CompressedTexImage3D(enum target, int level, enum internalformat, sizei width, sizei height, sizei depth, int border, sizei imageSize, const void *data)	–	–
CompressedTexSubImage1D(enum target, int level, int xoffset, sizei width, enum format, sizei imageSize, const void *data)	–	–
CompressedTexSubImage2D(enum target, int level, int xoffset, int yoffset, sizei width, sizei height, enum format, sizei imageSize, const void *data)	✓‡	✓‡
CompressedTexSubImage3D(enum target, int level, int xoffset, int yoffset, int zoffset, sizei width, sizei height, sizei depth, enum format, sizei imageSize, const void *data)	–	–
GetCompressedTexImage(enum target, int lod, void *img)	–	–
TexParameter{if}[v](enum target, enum pname, T param)		
target = TEXTURE_2D	✓	†
target = TEXTURE_1D,TEXTURE_3D,TEXTURE_CUBE_MAP	–	–
pname = TEXTURE_MIN_FILTER,TEXTURE_MAG_FILTER	✓	†
pname = TEXTURE_WRAP_S,TEXTURE_WRAP_T	✓	†
pname = TEXTURE_BORDER_COLOR	–	–
pname = TEXTURE_MIN_LOD,TEXTURE_MAX_LOD	–	–
pname = TEXTURE_BASE_LEVEL,TEXTURE_MAX_LEVEL	–	–
pname = TEXTURE_WRAP_R	–	–
pname = TEXTURE_LOD_BIAS	–	–
pname = DEPTH_TEXTURE_MODE	–	–
pname = TEXTURE_COMPARE_MODE	–	–
pname = TEXTURE_COMPARE_FUNC	–	–
pname = TEXTURE_PRIORITY	–	–
pname = GENERATE_MIPMAP	✓	†
GetTexParameter{if}v(enum target, enum pname, T *params)	✓	†
GetTexLevelParameter{if}v(enum target, int level, enum pname, T *params)	–	–
BindTexture(enum target, uint texture)		
target = TEXTURE_2D	✓	✓
target = TEXTURE_1D,TEXTURE_3D,TEXTURE_CUBE_MAP	–	–
DeleteTextures(sizei n, const uint *textures)	✓	✓
GenTextures(sizei n, uint *textures)	✓	✓
IsTexture(uint texture)	✓	✓
AreTexturesResident(sizei n, uint *textures, boolean *residences)	–	–
PrioritizeTextures(sizei n, uint *textures, clampf *priorities)	–	–
Enable/Disable(enum cap)		
cap = TEXTURE_2D	✓	✓

OpenGL 1.5	Common	Common-Lite
cap = TEXTURE_1D,TEXTURE_3D,TEXTURE_CUBE_MAP	–	–
TexEnv{if}[v](enum target, enum pname, T param)		
pname = TEXTURE_ENV_COLOR	✓	†
pname = TEXTURE_ENV_MODE:		
param = MODULATE,REPLACE,DECAL	✓	†
param = BLEND,ADD	✓	†
param = COMBINE	✓	†
pname = COMBINE_RGB,COMBINE_ALPHA	✓	†
pname = SRC{012}_RGB,SRC{012}_ALPHA	✓	†
pname = OPERAND{012}_RGB,OPERAND{012}_ALPHA	✓	†
pname = RGB_SCALE,ALPHA_SCALE	✓	†
GetTexEnv{if}v(enum target, enum pname, T *params)	✓	†

■ Texturing with 2D images is a critical feature for entertainment, presentation, and engineering applications. 1D, 3D, and cube map textures are less important. Texture objects are required for managing multiple textures. In some applications packing multiple textures into a single large texture is necessary for performance, therefore subimage support is also included. Copying from the framebuffer is useful for many shading algorithms. A limited set of formats, types and internal formats is included. The RGB component ordering is always RGB or RGBA rather than BGRA since there is no real perceived advantage to using BGRA. Format conversions for copying from the framebuffer are more liberal than for images specified in application memory, since an application usually has control over images authored as part of the application, but has little control over the framebuffer format. Unsupported **CopyTexture** conversions generate an INVALID_OPERATION error, since the error is dependent on the presence of a particular color component in the colorbuffer. This behavior parallels the error treatment for attempts to read from a non-existent depth or stencil buffer.

Texture borders are not included, since they are often not completely supported by full OpenGL implementations. All filter modes are supported since they represent a useful set of quality and speed options. Edge clamp and repeat wrap modes are both supported since these are most commonly used. Texture priorities are not supported since they are seldom used by applications. Similarly, the ability to control the LOD range and the base and maximum mipmap image levels is not included, since these features are used by a narrow set of applications. Since all of the supported texture parameters are scalar valued, the vector form of the parameter command is eliminated.

Auto mipmap generation has been added to OpenGL ES 1.1 since it can offload the application from having to generate mip-levels. Hardware implementations can potentially accelerate auto mip-level generation especially for video textures or when rendering to texture.

Compressed textures are important for reducing space and bandwidth requirements. The OpenGL 1.5 compression infrastructure is retained (for 2D textures) and a simple palette-based compression format is added as a required profile extension. ❑

3.8.6 Texture Environments and Texture Functions

All OpenGL 1.5 texture environments except for the texture crossbar are supported.

COMBINE_RGB	Texture Function
REPLACE	$Arg0$
MODULATE	$Arg0 * Arg1$
ADD	$Arg0 + Arg1$
ADD_SIGNED	$Arg0 + Arg1 - 0.5$
INTERPOLATE	$Arg0 * Arg2 + Arg1 * (1 - Arg2)$
SUBTRACT	$Arg0 - Arg1$
DOT3_RGB	$4 \times ((Arg0_r - 0.5) * (Arg1_r - 0.5) + (Arg0_g - 0.5) * (Arg1_g - 0.5) + (Arg0_b - 0.5) * (Arg1_b - 0.5))$
DOT3_RGBA	$4 \times ((Arg0_r - 0.5) * (Arg1_r - 0.5) + (Arg0_g - 0.5) * (Arg1_g - 0.5) + (Arg0_b - 0.5) * (Arg1_b - 0.5))$

COMBINE_ALPHA	Texture Function
REPLACE	$Arg0$
MODULATE	$Arg0 * Arg1$
ADD	$Arg0 + Arg1$
ADD_SIGNED	$Arg0 + Arg1 - 0.5$
INTERPOLATE	$Arg0 * Arg2 + Arg1 * (1 - Arg2)$
SUBTRACT	$Arg0 - Arg1$

Table 3.4: COMBINE texture functions. The scalar expression computed for the DOT3_RGB and DOT3_RGBA functions is placed into each of the 3 (RGB) or 4 (RGBA) components of the output. The result generated from COMBINE_ALPHA is ignored for DOT3_RGBA.

SRC*n*_RGB	OPERAND*n*_RGB	Argument
TEXTURE	SRC_COLOR	C_s
	ONE_MINUS_SRC_COLOR	$1 - C_s$
	SRC_ALPHA	A_s
	ONE_MINUS_SRC_ALPHA	$1 - A_s$
CONSTANT	SRC_COLOR	C_c
	ONE_MINUS_SRC_COLOR	$1 - C_c$
	SRC_ALPHA	Ac
	ONE_MINUS_SRC_ALPHA	$1 - A_c$
PRIMARY_COLOR	SRC_COLOR	C_f
	ONE_MINUS_SRC_COLOR	$1 - C_f$
	SRC_ALPHA	A_f
	ONE_MINUS_SRC_ALPHA	$1 - A_f$
PREVIOUS	SRC_COLOR	C_p
	ONE_MINUS_SRC_COLOR	$1 - C_p$
	SRC_ALPHA	A_p
	ONE_MINUS_SRC_ALPHA	$1 - A_p$

Table 3.5: Arguments for COMBINE_RGB functions.

SRC*n*_ALPHA	OPERAND*n*_ALPHA	Argument
TEXTURE	SRC_ALPHA	A_s
	ONE_MINUS_SRC_ALPHA	$1 - A_s$
CONSTANT	SRC_ALPHA	A_c
	ONE_MINUS_SRC_ALPHA	$1 - A_c$
PRIMARY_COLOR	SRC_ALPHA	A_f
	ONE_MINUS_SRC_ALPHA	$1 - A_f$
PREVIOUS	SRC_ALPHA	A_p
	ONE_MINUS_SRC_ALPHA	$1 - A_p$

Table 3.6: Arguments for COMBINE_ALPHA functions.

3.9 Fog

Fog is fully supported except for FOG_COORD_SRC and color index related modes.

OpenGL 1.5	**Common**	**Common-Lite**
Fogf[v](enum pname, T param)		
pname = FOG_MODE,FOG_DENSITY,FOG_START,FOG_END,FOG_COLOR	✓	†
pname = FOG_INDEX	–	–
pname = FOG_COORD_SRC	–	–
Fogi[v](enum pname, T param)	–	–
Enable/Disable(FOG)	✓	✓

■ Fog is useful for entertainment applications as a way to manage frame rate while hiding drawing mistakes. It can be emulated using texturing, but is often needed in applications that already use texturing for other purposes. Fog does present an implementation burden, but is used in enough applications to justify inclusion. Implementations can reduce the computational burden by computing fog values at each vertex rather than each pixel. ❑

Chapter 4

Per-Fragment Operations and the Framebuffer

4.1 Per-Fragment Operations

All OpenGL 1.5 per-fragment operations are supported, except for color index related operations and certain advanced blending features (**BlendColor**, **BlendEquation**, and blend squaring). Depth and stencil operations are supported, but a selected config is not required to include a depth or stencil buffer.

OpenGL 1.5	Common	Common-Lite
Enable/Disable (SCISSOR_TEST)	✓	✓
Scissor (int x, int y, sizei width, sizei height)	✓	✓
Enable/Disable (SAMPLE_COVERAGE)	✓	✓
Enable/Disable (SAMPLE_ALPHA_TO_COVERAGE)	✓	✓
Enable/Disable (SAMPLE_ALPHA_TO_ONE)	✓	✓
SampleCoverage (clampf value, boolean invert)	✓	†
Enable/Disable (ALPHA_TEST)	✓	✓
AlphaFunc (enum func, clampf ref)	✓	†
Enable/Disable (STENCIL_TEST)	✓	✓
StencilFunc (enum func, int ref, uint mask)	✓	✓
StencilOp (enum fail, enum zfail, enum zpass)		
fail, zfail, zpass = KEEP	✓	✓
fail, zfail, zpass = ZERO	✓	✓
fail, zfail, zpass = REPLACE	✓	✓
fail, zfail, zpass = INCR	✓	✓
fail, zfail, zpass = DECR	✓	✓
fail, zfail, zpass = INVERT	✓	✓
fail, zfail, zpass = INCR_WRAP	–	–
fail, zfail, zpass = DECR_WRAP	–	–

OpenGL 1.5	Common	Common-Lite
Enable/Disable (DEPTH_TEST)	✓	✓
DepthFunc (enum func)	✓	✓
Enable/Disable (BLEND)	✓	✓
BlendFunc (enum sfactor, enum dfactor)	✓	✓
BlendFuncSeparate (enum srcRGB, enum dstRGB, enum srcAlpha, enum dstAlpha)	–	–
BlendEquation (enum mode)	–	–
BlendColor (clampf red, clampf green, clampf blue, clampf alpha)	–	–
Enable/Disable (DITHER)	✓	✓
Enable/Disable (INDEX_LOGIC_OP)	–	–
Enable/Disable (COLOR_LOGIC_OP)	✓	✓
LogicOp (enum opcode)	✓	✓
BeginQuery (enum target, uint id)	–	–
EndQuery (enum target)	–	–
GenQueries (sizei n, uint *ids)	–	–
DeleteQueries (sizei n, uint *ids)	–	–

4.1.1 Blending

The **BlendFuncSeparate**, **BlendEquation** and **BlendColor** commands defined in the OpenGL 1.5 specification are not supported by OpenGL ES 1.1. Only **BlendFunc** is supported. **BlendFunc** *src* indicates how to compute a source blending factor, while *dst* indicates how to compute a destination factor. The possible arguments and their corresponding computed source and destination factors are summarized in tables 4.2 and 4.3.

Value	Blend Factors
ZERO	$(0, 0, 0, 0)$
ONE	$(1, 1, 1, 1)$
DST_COLOR	(R_d, G_d, B_d, A_d)
ONE_MINUS_DST_COLOR	$(1, 1, 1, 1) - (R_d, G_d, B_d, A_d)$
SRC_ALPHA	(A_s, A_s, A_s, A_s)
ONE_MINUS_SRC_ALPHA	$(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)$
DST_ALPHA	(A_d, A_d, A_d, A_d)
ONE_MINUS_DST_ALPHA	$(1, 1, 1, 1) - (A_d, A_d, A_d, A_d)$
SRC_ALPHA_SATURATE	$(f, f, f, 1)$

Table 4.2: Values controlling the source blending function and the source blending values they compute. $f = min(A_s, 1 - A_d)$.

Value	Blend Factors
ZERO	$(0, 0, 0, 0)$
ONE	$(1, 1, 1, 1)$
SRC_COLOR	(R_s, G_s, B_s, A_s)
ONE_MINUS_SRC_COLOR	$(1, 1, 1, 1) - (R_s, G_s, B_s, A_s)$
SRC_ALPHA	(A_s, A_s, A_s, A_s)
ONE_MINUS_SRC_ALPHA	$(1, 1, 1, 1) - (A_s, A_s, A_s, A_s)$
DST_ALPHA	(A_d, A_d, A_d, A_d)
ONE_MINUS_DST_ALPHA	$(1, 1, 1, 1) - (A_d, A_d, A_d, A_d)$

Table 4.3: Values controlling the destination blending function and the destination blending values they compute.

■ Scissor is useful for providing complete control over where pixels are drawn and some form of window/drawing-surface scissoring is typically present in most rasterizers so the cost is small. Alpha testing is useful for early rejection of transparent pixels and for some kinds of keying. Stenciling is useful for drawing with masks and for a number of presentation effects and an implementation is not required to support a stencil buffer (just the API and the correct behavior when not present). Depth buffering is essential for many 3D applications and the profile should require some form of depth buffer to be present. Blending is necessary for implementing transparency, color sums, and some other useful rendering effects. Dithering is useful on displays with low color resolution, and the inclusion doesn't require dithering to be implemented in the renderer. Logic op is useful for efficient highlighting operations. Masked operations are supported since they are often used in more complex

operations and are needed to achieve invariance. Support for blend equations other than add and blend color would be useful, but are only included in the Imaging Subset of OpenGL 1.3 and are therefore not included. In addition, the **BlendFunc** arguments are the same as defined in the OpenGL 1.3 specification. Changes to the **BlendFunc** functionality described in the OpenGL 1.5 specification are not included. ❑

4.2 Whole Framebuffer Operations

All whole framebuffer operations are supported except for color index related operations, drawing to different color buffers, and accumulation buffer.

OpenGL 1.5	Common	Common-Lite
DrawBuffer (`enum mode`)	–	–
IndexMask (`uint mask`)	–	–
ColorMask (`boolean red, boolean green, boolean blue, boolean alpha`)	✓	✓
DepthMask (`boolean flag`)	✓	✓
StencilMask (`uint mask`)	✓	✓
Clear (`bitfield mask`)	✓	✓
ClearColor (`clampf red, clampf green, clampf blue, clampf alpha`)	✓	†
ClearIndex (`float c`)	–	–
ClearDepth (`clampd depth`)	♢	†
ClearStencil (`int s`)	✓	✓
ClearAccum (`float red, float green, float blue, float alpha`)	–	–
Accum (`enum op, float value`)	–	–

■ Multiple drawing buffers are not exposed; an application can only draw to the default buffer, so **DrawBuffer** is not necessary. The accumulation buffer is not used in many applications, though it is useful as a non-interactive antialiasing technique. ❑

4.3 Drawing, Reading, and Copying Pixels

ReadPixels is supported with the following exceptions: the depth and stencil buffers cannot be read from and the number of format and type combinations for **ReadPixels** is severely restricted. Two format/type combinations are supported: format `RGBA` and type `UNSIGNED_BYTE` for portability; and one implementation-specific format/type combination queried using the tokens `IMPLEMENTATION_COLOR_READ_FORMAT_OES` and `IMPLEMENTATION_COLOR_READ_TYPE_OES` (`OES_read_format` extension). The format and type combinations that can be returned from these queries are listed in Table 3.1. **CopyPixels** and **ReadBuffer** are not supported. Read operations return data from the default color buffer.

OpenGL 1.5	Common	Common-Lite
ReadBuffer (`enum mode`)	–	–

OpenGL 1.5	Common	Common-Lite
ReadPixels(`int x, int y,sizei width, sizei height, enum format, enum type, void *pixels`)	✓‡	✓‡
CopyPixels(`int x, int y, sizei width, sizei height, enum type`)	–	–

■ Reading the color buffer is useful for some applications and also provides a platform independent method for testing. The inclusion of the `OES_read_format` extension allows an implementation to support a more efficient format without increasing the number of formats that must be supported. Pixel copies can be implemented by reading to the host and then drawing to the color buffer (using texture mapping for the drawing part). Image copy performance is important to many presentation applications, so **CopyPixels** may be revisited in a future revision. Drawing to and reading from the depth and stencil buffers is not used frequently in applications (though it would be convenient for testing), so it is not included. **ReadBuffer** is not required since the concept of multiple drawing buffers is not exposed. ❑

Chapter 5

Special Functions

5.1 Evaluators

Evaluators are not supported.

OpenGL 1.5	Common	Common-Lite
Map1{**fd**}`(enum target, T u1, T u2, int stride, int order, T points)`	–	–
Map2{**fd**}`(enum target, T u1, T u2, int ustride, int uorder, T v1, T v2, int vstride, int vorder, T *points)`	–	–
GetMap{**ifd**}**v**`(enum target, enum query, T *v)`	–	–
EvalCoord{**12**}{**fd**}**[v]**`(T coord)`	–	–
MapGrid1{**fd**}`(int un, T u1, T u2)`	–	–
MapGrid2{**fd**}`(int un, T u1, T u2, T v1, T v2)`	–	–
EvalMesh1`(enum mode, int i1, int i2)`	–	–
EvalMesh2`(enum mode, int i1, int i2, int j1, int j2)`	–	–
EvalPoint1`(int i)`	–	–
EvalPoint2`(int i, int j)`	–	–

■ Evaluators are not used by many applications other than sophisticated CAD applications. ❑

5.2 Selection

Selection is not supported.

OpenGL 1.5	Common	Common-Lite
InitNames`(void)`	–	–
LoadName`(uint name)`	–	–
PushName`(uint name)`	–	–
PopName`(void)`	–	–
RenderMode`(enum mode)`	–	–
SelectBuffer`(sizei size, uint *buffer)`	–	–

■ Selection is not used by many applications. There are other methods that applications can use to implement picking operations. ❑

5.3 Feedback

Feedback is not supported.

OpenGL 1.5	Common	Common-Lite
FeedbackBuffer (sizei size, enum type, float *buffer)	–	–
PassThrough (float token)	–	–

■ Feedback is seldom used. ❑

5.4 Display Lists

Display lists are not supported.

OpenGL 1.5	Common	Common-Lite
NewList (uint list, enum mode)	–	–
EndList (void)	–	–
CallList (uint list)	–	–
CallLists (sizei n, enum type, const void *lists)	–	–
ListBase (uint base)	–	–
GenLists (sizei range)	–	–
IsList (uint list)	–	–
DeleteLists (uint list, sizei range)	–	–

■ Display lists are used by many applications — sometimes to achieve better performance and sometimes for convenience. The implementation complexity associated with display lists is too large for the implementation targets envisioned for this profile. ❑

5.5 Flush and Finish

Flush and **Finish** are supported.

OpenGL 1.5	Common	Common-Lite
Flush (void)	✓	✓
Finish (void)	✓	✓

■ Applications need some manner to guarantee rendering has completed, so **Finish** needs to be supported. **Flush** can be trivially supported. ❑

5.6 Hints

Hints are retained except for the hints relating to the unsupported polygon smoothing and compression of textures (including retrieving compressed textures) features.

OpenGL 1.5	**Common**	**Common-Lite**
Hint(enum target, enum mode)		
target = PERSPECTIVE_CORRECTION_HINT	✓	✓
target = POINT_SMOOTH_HINT	✓	✓
target = LINE_SMOOTH_HINT	✓	✓
target = FOG_HINT	✓	✓
target = TEXTURE_COMPRESSION_HINT	–	–
target = POLYGON_SMOOTH_HINT	–	–
target = GENERATE_MIPMAP_HINT	✓	✓

■ Applications and implementations still need some method for expressing permissible speed versus quality trade-offs. The implementation cost is minimal. There is no value in retaining the hints for unsupported features. ❑

Chapter 6

State and State Requests

6.1 Querying GL State

State queries are supported for *static* and *dynamic* state explicitly supported in the profile. The supported GL state queries can be categorized into simple queries, enumerated queries, texture queries, pointer and string queries, and buffer object queries.

The values of the strings returned by **GetString** are specified as part of the profile definition. In particular, the version string indicates the particular OpenGL ES profile as well as the version of that profile. Strings are listed in Table 6.1.

As the profile is revised, the `VERSION` string is updated to indicate the revision. The string format is fixed and includes a two-character profile identifier: `CM` for the Common and `CL` for the Common-Lite profile; and the two-digit version number (`X.Y`).

Strings	
`VENDOR`	as defined by OpenGL 1.5
`RENDERER`	as defined by OpenGL 1.5
`VERSION`	"`OpenGL ES-XX 1.1`" `XX={CM,CL}`
`EXTENSIONS`	as defined by OpenGL 1.5

Table 6.1: String State

Client and server attribute stacks are not supported by the profiles; consequently, the commands **PushAttrib**, **PopAttrib**, **PushClientAttrib**, and **PopClientAttrib** are not supported. Gets are supported by the profiles to allow an application to save and restore dynamic state.

OpenGL 1.5	Common	Common-Lite
GetBooleanv(enum pname, boolean *params)	✓	✓
GetIntegerv(enum pname, int *params)	✓	✓
GetFloatv(enum pname, float *params)	✓	†
GetDoublev(enum pname, double *params)	–	–
IsEnabled(enum cap)	✓	✓
GetClipPlane(enum plane, double eqn[4])	–	–
GetClipPlanef(enum plane, float eqn[4])	♢	†
GetLightfv(enum light, enum pname, float *params)	✓	†
GetLightiv(enum light, enum pname, int *params)	–	–
GetMaterialfv(enum face, enum pname, float *params)	✓	†
GetMaterialiv(enum face, enum pname, int *params)	–	–
GetTexEnv{**if**}**v**(enum env, enum pname, T *params)	✓	†
GetTexGen{**ifd**}**v**(enum env, enum pname, T *params)	–	–
GetTexParameter{**if**}**v**(enum target, enum pname, T *params)	✓	†
GetTexLevelParameter{**if**}**v**(enum target, int lod, enum pname, T *params)	–	–
GetPixelMap{**ui us f**}**v**(enum map, T data)	–	–
GetMap{**ifd**}**v**(enum map, enum value, T data)	–	–
GetBufferParameteriv(enum target, enum pname, boolean *params)	✓	✓
GetTexImage(enum tex, int lod, enum format, enum type, void *img)	–	–
GetCompressedTexImage(enum tex, int lod, void *img)	–	–
IsTexture(uint texture)	✓	✓
GetPolygonStipple(void *pattern)	–	–
GetColorTable(enum target, enum format, enum type, void *table)	–	–
GetColorTableParameter{**if**}**v**(enum target, enum pname, T params)	–	–
GetPointerv(enum pname, void **params)	✓	✓
GetString(enum name)	✓	✓
IsQuery(uint id)	–	–
GetQueryiv(enum target, enum pname, int *params)	–	–
GetQueryObjectiv(uint id, enum pname, int *params)	–	–
GetQueryObjectuiv(uint id, enum pname, uint *params)	–	–
IsBuffer(uint buffer)	✓	✓
GetBufferSubData(enum target, intptr offset, sizeiptr size, void *data)	–	–
GetBufferPointerv(enum target, enum pname, void **params)	–	–

OpenGL 1.5	Common	Common-Lite
PushAttrib (bitfield mask)	–	–
PopAttrib (void)	–	–
PushClientAttrib (bitfield mask)	–	–
PopClientAttrib (void)	–	–

■ There are several reasons why one type or another of internal state needs to be queried by an application. The application may need to dynamically discover implementation limits (pixel component sizes, texture dimensions, etc.), or the application might be part of a layered library and it may need to save and restore any state that it disturbs as part of its rendering. **PushAttrib** and **PopAttrib** can be used to perform this but they are expensive to implement in hardware since we need an attribute stack depth greater than 1. An attribute stack depth of 4 was proposed but was rejected because an application would still have to handle stack overflow which was considered unacceptable. Gets can be efficiently implemented if the implementation shadows states on the CPU. Gets also allow an infinite stack depth so an application will never have to worry about stack overflow errors. The string queries are retained as they provide important versioning, and extension information. ❑

6.2 State Tables

The following tables summarize state that is present in the Common and Common-Lite profiles. The tables also indicate which state variables are obtained with what commands. State variables that can be obtained using any of GetBooleanv, GetIntegerv, or GetFloatv are listed with just one of these commands - the one that is most appropriate given the type of data to be returned and the profile used. These state variables cannot be obtained using IsEnabled. However, state variables for which IsEnabled is listed as the query command can also be obtained using GetBooleanv, GetIntegerv, and GetFloatv. State variables for which any other command is listed as the query command can be obtained only by using that command.

State appearing in *italic* indicates unnamed state. All state has initial values identical to those specified in OpenGL 1.5.

State	Exposed	Queriable	Common Get	Common-Lite Get
Begin/end object	–	–	–	–
Previous line vertex	✓	–	–	–
First line-vertex flag	✓	–	–	–
First vertex of line loop	✓	–	–	–
Line stipple counter	–	–	–	–
Polygon vertices	–	–	–	–
Number of polygon vertices	–	–	–	–
Previous two triangle strip vertices	✓	–	–	–
Number of triangle strip vertices	✓	–	–	–
Triangle strip A/B pointer	✓	–	–	–
Quad vertices	–	–	–	–
Number of quad strip vertices	–	–	–	–

Table 6.4: GL Internal begin-end state variables

State	Exposed	Queriable	Common Get	Common-Lite Get
CURRENT_COLOR	✓	✓	**GetIntegerv GetFloatv**	**GetIntegerv GetFixedv**
CURRENT_INDEX	–	–	–	–
CURRENT_TEXTURE_COORDS	✓	✓	**GetFloatv**	**GetFixedv**
CURRENT_NORMAL	✓	✓	**GetFloatv**	**GetFixedv**
Color associated with last vertex	✓	–	–	–
Color index associated with last vertex	–	–	–	–
Texture coordinates associated with last vertex	✓	–	–	–
CURRENT_RASTER_POSITION	–	–	–	–
CURRENT_RASTER_DISTANCE	–	–	–	–
CURRENT_RASTER_COLOR	–	–	–	–
CURERNT_RASTER_INDEX	–	–	–	–
CURRENT_RASTER_TEXTURE_COORDS	–	–	–	–
CURRENT_RASTER_POSITION_VALID	–	–	–	–
EDGE_FLAG	–	–	–	

Table 6.5: Current Values and Associated Data

State	Exposed	Queriable	Common Get	Common-Lite Get
CLIENT_ACTIVE_TEXTURE	✓	✓	**GetIntegerv**	**GetIntegerv**
VERTEX_ARRAY	✓	✓	**IsEnabled**	**IsEnabled**
VERTEX_ARRAY_SIZE	✓	✓	**GetIntegerv**	**GetIntegerv**
VERTEX_ARRAY_STRIDE	✓	✓	**GetIntegerv**	**GetIntegerv**
VERTEX_ARRAY_TYPE	✓	✓	**GetIntegerv**	**GetIntegerv**
VERTEX_ARRAY_POINTER	✓	✓	**GetPointerv**	**GetPointerv**
NORMAL_ARRAY	✓	✓	**IsEnabled**	**IsEnabled**
NORMAL_ARRAY_STRIDE	✓	✓	**GetIntegerv**	**GetIntegerv**
NORMAL_ARRAY_TYPE	✓	✓	**GetIntegerv**	**GetIntegerv**
NORMAL_ARRAY_POINTER	✓	✓	**GetPointerv**	**GetPointerv**
FOG_COORD_ARRAY	–	–	–	–
FOG_COORD_ARRAY_STRIDE	–	–	–	–
FOG_COORD_ARRAY_TYPE	–	–	–	–
FOG_COORD_ARRAY_POINTER	–	–	–	–
COLOR_ARRAY	✓	✓	**IsEnabled**	**IsEnabled**
COLOR_ARRAY_SIZE	✓	✓	**GetIntegerv**	**GetIntegerv**
COLOR_ARRAY_STRIDE	✓	✓	**GetIntegerv**	**GetIntegerv**
COLOR_ARRAY_TYPE	✓	✓	**GetIntegerv**	**GetIntegerv**
COLOR_ARRAY_POINTER	✓	✓	**GetPointerv**	**GetPointerv**
SECONDARY_COLOR_ARRAY	–	–	–	–
SECONDARY_COLOR_ARRAY_SIZE	–	–	–	–
SECONDARY_COLOR_ARRAY_STRIDE	–	–	–	–
SECONDARY_COLOR_ARRAY_TYPE	–	–	–	–
SECONDARY_COLOR_ARRAY_POINTER	–	–	–	–
INDEX_ARRAY	–	–	–	–
INDEX_ARRAY_STRIDE	–	–	–	–
INDEX_ARRAY_TYPE	–	–	–	–
INDEX_ARRAY_POINTER	–	–	–	–
TEXTURE_COORD_ARRAY	✓	✓	**IsEnabled**	**IsEnabled**
TEXTURE_COORD_ARRAY_SIZE	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_COORD_ARRAY_STRIDE	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_COORD_ARRAY_TYPE	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_COORD_ARRAY_POINTER	✓	✓	**GetPointerv**	**GetPointerv**
EDGE_FLAG_ARRAY	–	–	–	–
EDGE_FLAG_ARRAY_STRIDE	–	–	–	–
EDGE_FLAG_ARRAY_POINTER	–	–	–	–
ARRAY_BUFFER_BINDING	✓	✓	**GetIntegerv**	**GetIntegerv**
VERTEX_ARRAY_BUFFER_BINDING	✓	✓	**GetIntegerv**	**GetIntegerv**
NORMAL_ARRAY_BUFFER_BINDING	✓	✓	**GetIntegerv**	**GetIntegerv**
FOG_COORD_ARRAY_BUFFER_BINDING	–	–	–	–
COLOR_ARRAY_BUFFER_BINDING	✓	✓	**GetIntegerv**	**GetIntegerv**
SECONDARY_COLOR_ARRAY_BUFFER_BINDING	–	–	–	–
TEXTURE_COORD_ARRAY_BUFFER_BINDING	✓	✓	**GetIntegerv**	**GetIntegerv**
ELEMENT_ARRAY_BUFFER_BINDING	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 6.6: Vertex Array Data

State	Exposed	Queriable	Common Get	Common-Lite Get
BUFFER_SIZE	✓	✓	**GetBufferParameteriv**	**GetBufferParameteriv**
BUFFER_USAGE	✓	✓	**GetBufferParameteriv**	**GetBufferParameteriv**
BUFFER_ACCESS	–	–	–	–
BUFFER_MAPPED	–	–	–	–
BUFFER_MAP_POINTER	–	–	–	–

Table 6.7: Buffer Object State

State	Exposed	Queriable	Common Get	Common-Lite Get
COLOR_MATRIX	–	–	–	–
MODELVIEW_MATRIX	✓	✓	**GetFloatv**	**GetFixedv**
PROJECTION_MATRIX	✓	✓	**GetFloatv**	**GetFixedv**
TEXTURE_MATRIX	✓	✓	**GetFloatv**	**GetFixedv**
VIEWPORT	✓	✓	**GetIntegerv**	**GetIntegerv**
DEPTH_RANGE	✓	✓	**GetFloatv**	**GetFixedv**
COLOR_MATRIX_STACK_DEPTH	–	–	–	–
MODELVIEW_STACK_DEPTH	✓	✓	**GetIntegerv**	**GetIntegerv**
PROJECTION_STACK_DEPTH	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_STACK_DEPTH	✓	✓	**GetIntegerv**	**GetIntegerv**
MATRIX_MODE	✓	✓	**GetIntegerv**	**GetIntegerv**
NORMALIZE	✓	✓	**IsEnabled**	**IsEnabled**
RESCALE_NORMAL	✓	✓	**IsEnabled**	**IsEnabled**
CLIP_PLANE{0-5}	✓	✓	**GetClipPlanef**	**GetClipPlanex**
CLIP_PLANE{0-5}	✓	✓	**IsEnabled**	**IsEnabled**

Table 6.8: Transformation State

State	Exposed	Queriable	Common Get	Common-Lite Get
FOG_COLOR	✓	✓	**GetFloatv**	**GetFixedv**
FOG_INDEX	–	–	–	–
FOG_DENSITY	✓	✓	**GetFloatv**	**GetFixedv**
FOG_START	✓	✓	**GetFloatv**	**GetFixedv**
FOG_END	✓	✓	**GetFloatv**	**GetFixedv**
FOG_MODE	✓	✓	**GetIntegerv**	**GetIntegerv**
FOG	✓	✓	**IsEnabled**	**IsEnabled**
SHADE_MODEL	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 6.9: Coloring

State	Exposed	Queriable	Common Get	Common-Lite Get
LIGHTING	✓	✓	**IsEnabled**	**IsEnabled**
COLOR_MATERIAL	✓	✓	**Is Enabled**	**IsEnabled**
COLOR_MATERIAL_PARAMETER	–	–	–	–
COLOR_MATERIAL_FACE	–	–	–	–
AMBIENT (material)	✓	✓	**GetMaterialfv**	**GetMaterialxv**
DIFFUSE (material)	✓	✓	**GetMaterialfv**	**GetMaterialxv**
SPECULAR (material)	✓	✓	**GetMaterialfv**	**GetMaterialxv**
EMISSION (material)	✓	✓	**GetMaterialfv**	**GetMaterialxv**
SHININESS (material)	✓	✓	**GetMaterialfv**	**GetMaterialxv**
LIGHT_MODEL_AMBIENT	✓	✓	**GetFloatv**	**GetFixedv**
LIGHT_MODEL_LOCAL_VIEWER	–	–	–	–
LIGHT_MODEL_TWO_SIDE	✓	✓	**GetBooleanv**	**GetBooleanv**
LIGHT_MODEL_COLOR_CONTROL	–	–	–	–
AMBIENT ($light_i$)	✓	✓	**GetLightfv**	**GetLightxv**
DIFFUSE ($light_i$)	✓	✓	**GetLightfv**	**GetLightxv**
SPECULAR ($light_i$)	✓	✓	**GetLightfv**	**GetLightxv**
POSITION ($light_i$)	✓	✓	**GetLightfv**	**GetLightxv**
CONSTANT_ATTENUATION	✓	✓	**GetLightfv**	**GetLightxv**
LINEAR_ATTENUATION	✓	✓	**GetLightfv**	**GetLightxv**
QUADRATIC_ATTENUATION	✓	✓	**GetLightfv**	**GetLightxv**
SPOT_DIRECTION	✓	✓	**GetLightfv**	**GetLightxv**
SPOT_EXPONENT	✓	✓	**GetLightfv**	**GetLightxv**
SPOT_CUTOFF	✓	✓	**GetLightfv**	**GetLightxv**
LIGHT{0-7}	✓	✓	**IsEnabled**	**IsEnabled**
COLOR_INDEXES	–	–	–	–

Table 6.10: Lighting

State	Exposed	Queriable	Common Get	Common-Lite Get
POINT_SIZE	✓	✓	**GetFloatv**	**GetFixedv**
POINT_SMOOTH	✓	✓	**IsEnabled**	**IsEnabled**
POINT_SIZE_MIN	✓	✓	**GetFloatv**	**GetFixedv**
POINT_SIZE_MAX	✓	✓	**GetFloatv**	**GetFixedv**
POINT_FADE_THRESHOLD_SIZE	✓	✓	**GetFloatv**	**GetFixedv**
POINT_DISTANCE_ATTENUATION	✓	✓	**GetFloatv**	**GetFixedv**
LINE_WIDTH	✓	✓	**GetFloatv**	**GetFixedv**
LINE_SMOOTH	✓	✓	**IsEnabled**	**IsEnabled**
LINE_STIPPLE_PATTERN	–	–	–	–
LINE_STIPPLE_REPEAT	–	–	–	–
LINE_STIPPLE	–	–	–	–
CULL_FACE	✓	✓	**IsEnabled**	**IsEnabled**
CULL_FACE_MODE	✓	✓	**GetIntegerv**	**GetIntegerv**
FRONT_FACE	✓	✓	**GetIntegerv**	**GetIntegerv**
POLYGON_SMOOTH	–	–	–	–
POLYGON_MODE	–	–	–	–
POLYGON_OFFSET_FACTOR	✓	✓	**GetFloatv**	**GetFixedv**
POLYGON_OFFSET_UNITS	✓	✓	**GetFloatv**	**GetFixedv**
POLYGON_OFFSET_POINT	–	–	–	–
POLYGON_OFFSET_LINE	–	–	–	–
POLYGON_OFFSET_FILL	✓	✓	**IsEnabled**	**IsEnabled**
POLYGON_STIPPLE	–	–	–	–

Table 6.11: Rasterization

State	Exposed	Queriable	Common Get	Common-Lite Get
MULTISAMPLE	✓	✓	**IsEnabled**	**IsEnabled**
SAMPLE_ALPHA_TO_COVERAGE	✓	✓	**IsEnabled**	**IsEnabled**
SAMPLE_ALPHA_TO_ONE	✓	✓	**IsEnabled**	**IsEnabled**
SAMPLE_COVERAGE	✓	✓	**IsEnabled**	**IsEnabled**
SAMPLE_COVERAGE_VALUE	✓	✓	**GetFloatv**	**GetFixedv**
SAMPLE_COVERAGE_INVERT	✓	✓	**GetBooleanv**	**GetBooleanv**

Table 6.12: Multisampling

State	Exposed	Queriable	Common Get	Common-Lite Get
TEXTURE_1D	–	–	–	–
TEXTURE_2D	✓	✓	**IsEnabled**	**IsEnabled**
TEXTURE_3D	–	–	–	–
TEXTURE_CUBE_MAP	–	–	–	–
TEXTURE_BINDING_1D	–	–	–	–
TEXTURE_BINDING_2D	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_BINDING_3D	–	–	–	–
TEXTURE_BINDING_CUBE_MAP	–	–	–	
TEXTURE_CUBE_MAP_POSITIVE_X	–	–	–	–
TEXTURE_CUBE_MAP_NEGATIVE_X	–	–	–	–
TEXTURE_CUBE_MAP_POSITIVE_Y	–	–	–	–
TEXTURE_CUBE_MAP_NEGATIVE_Y	–	–	–	–
TEXTURE_CUBE_MAP_POSITIVE_Z	–	–	–	–
TEXTURE_CUBE_MAP_NEGATIVE_Z	–	–	–	–
TEXTURE_WIDTH	✓	–	–	–
TEXTURE_HEIGHT	✓	–	–	–
TEXTURE_DEPTH	–	–	–	–
TEXTURE_BORDER	–	–	–	–
TEXTURE_INTERNAL_FORMAT	✓	–	–	–
TEXTURE_RED_SIZE	✓	–	–	–
TEXTURE_GREEN_SIZE	✓	–	–	–
TEXTURE_BLUE_SIZE	✓	–	–	–
TEXTURE_ALPHA_SIZE	✓	–	–	–
TEXTURE_LUMINANCE_SIZE	✓	–	–	–
TEXTURE_INTENSITY_SIZE	–	–	–	–
TEXTURE_COMPRESSED	✓	–	–	–
TEXTURE_COMPRESSED_IMAGE_SIZE	✓	–	–	–
TEXTURE_BORDER_COLOR	–	–	–	–
TEXTURE_MIN_FILTER	✓	✓	**GetTexParameteriv**	**GetTexParameteriv**
TEXTURE_MAG_FILTER	✓	✓	**GetTexParameteriv**	**GetTexParameteriv**
TEXTURE_WRAP_S	✓	✓	**GetTexParameteriv**	**GetTexParameteriv**
TEXTURE_WRAP_T	✓	✓	**GetTexParameteriv**	**GetTexParameteriv**
TEXTURE_WRAP_R	–	–	–	–
TEXTURE_PRIORITY	–	–	–	–
TEXTURE_RESIDENT	–	–	–	–
TEXTURE_MIN_LOD	–	–	–	–
TEXTURE_MAX_LOD	–	–	–	–
TEXTURE_BASE_LEVEL	–	–	–	–
TEXTURE_MAX_LEVEL	–	–	–	–
GENERATE_MIPMAP	✓	✓	**GetTexParameteriv**	**GetTexParameteriv**

Table 6.13: Texture Objects

State	Exposed	Queriable	Common Get	Common-Lite Get
ACTIVE_TEXTURE	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_ENV_MODE	✓	✓	**GetTexEnviv**	**GetTexEnviv**
TEXTURE_ENV_COLOR	✓	✓	**GetTexEnvfv**	**GetTexEnvxv**
TEXTURE_GEN_{STRQ}	–	–	–	–
EYE_PLANE	–	–	–	–
OBJECT_PLANE	–	–	–	–
TEXTURE_GEN_MODE	–	–	–	–
COMBINE_RGB	✓	✓	**GetTexEnviv**	**GetTexEnviv**
COMBINE_ALPHA	✓	✓	**GetTexEnviv**	**GetTexEnviv**
SRC{012}_RGB	✓	✓	**GetTexEnviv**	**GetTexEnviv**
SRC{012}_ALPHA	✓	✓	**GetTexEnviv**	**GetTexEnviv**
OPERAND{012}_RGB	✓	✓	**GetTexEnviv**	**GetTexEnviv**
OPERAND{012}_ALPHA	✓	✓	**GetTexEnviv**	**GetTexEnviv**
RGB_SCALE	✓	✓	**GetTexEnviv**	**GetTexEnviv**
ALPHA_SCALE	✓	✓	**GetTexEnviv**	**GetTexEnviv**

Table 6.14: Texture Environment and Generation

State	Exposed	Queriable	Common Get	Common-Lite Get
DRAW_BUFFER	–	–	–	–
INDEX_WRITEMASK	–	–	–	–
COLOR_WRITEMASK	✓	✓	**GetBooleanv**	**GetBooleanv**
DEPTH_WRITEMASK	✓	✓	**GetBooleanv**	**GetBooleanv**
STENCIL_WRITEMASK	✓	✓	**GetIntegerv**	**GetIntegerv**
COLOR_CLEAR_VALUE	✓	✓	**GetFloatv**	**GetFixedv**
INDEX_CLEAR_VALUE	–	–	–	–
DEPTH_CLEAR_VALUE	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_CLEAR_VALUE	✓	✓	**GetIntegerv**	**GetIntegerv**
ACCUM_CLEAR_VALUE	–	–	–	–

Table 6.15: Framebuffer Control

State	Exposed	Queriable	Common Get	Common-Lite Get
SCISSOR_TEST	✓	✓	**IsEnabled**	**IsEnabled**
SCISSOR_BOX	✓	✓	**GetIntegerv**	**GetIntegerv**
ALPHA_TEST	✓	✓	**IsEnabled**	**IsEnabled**
ALPHA_TEST_FUNC	✓	✓	**GetIntegerv**	**GetIntegerv**
ALPHA_TEST_REF	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_TEST	✓	✓	**IsEnabled**	**IsEnabled**
STENCIL_FUNC	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_VALUE_MASK	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_REF	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_FAIL	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_PASS_DEPTH_FAIL	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_PASS_DEPTH_PASS	✓	✓	**GetIntegerv**	**GetIntegerv**
DEPTH_TEST	✓	✓	**IsEnabled**	**IsEnabled**
DEPTH_FUNC	✓	✓	**GetIntegerv**	**GetIntegerv**
BLEND	✓	✓	**IsEnabled**	**IsEnabled**
BLEND_SRC	✓	✓	**GetIntegerv**	**GetIntegerv**
BLEND_DST	✓	✓	**GetIntegerv**	**GetIntegerv**
BLEND_EQUATION	–	–	–	–
BLEND_COLOR	–	–	–	–
DITHER	✓	✓	**IsEnabled**	**IsEnabled**
INDEX_LOGIC_OP	–	–	–	–
COLOR_LOGIC_OP	✓	✓	**IsEnabled**	**IsEnabled**
LOGIC_OP_MODE	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 6.16: Pixel Operations

State	Exposed	Queriable	Common Get	Common-Lite Get
UNPACK_SWAP_BYTES	–	–	–	–
UNPACK_LSB_FIRST	–	–	–	–
UNPACK_IMAGE_HEIGHT	–	–	–	–
UNPACK_SKIP_IMAGES	–	–	–	–
UNPACK_ROW_LENGTH	–	–	–	–
UNPACK_SKIP_ROWS	–	–	–	–
UNPACK_SKIP_PIXELS	–	–	–	–
UNPACK_ALIGNMENT	✓	✓	**GetIntegerv**	**GetIntegerv**
PACK_SWAP_BYTES	–	–	–	–
PACK_LSB_FIRST	–	–	–	–
PACK_IMAGE_HEIGHT	–	–	–	–
PACK_SKIP_IMAGES	–	–	–	–
PACK_ROW_LENGTH	–	–	–	–
PACK_SKIP_ROWS	–	–	–	–
PACK_SKIP_PIXELS	–	–	–	–
PACK_ALIGNMENT	✓	✓	**GetIntegerv**	**GetIntegerv**
MAP_COLOR	–	–	–	–
MAP_STENCIL	–	–	–	–
INDEX_SHIFT	–	–	–	–
INDEX_OFFSET	–	–	–	–
RED_SCALE	–	–	–	–
GREEN_SCALE	–	–	–	–
BLUE_SCALE	–	–	–	–
ALPHA_SCALE	–	–	–	–
DEPTH_SCALE	–	–	–	–
RED_BIAS	–	–	–	–
GREEN_BIAS	–	–	–	–
BLUE_BIAS	–	–	–	–
ALPHA_BIAS	–	–	–	–
DEPTH_BIAS	–	–	–	–

Table 6.17: Pixels

State	Exposed	Queriable	Common Get	Common-Lite Get
COLOR_TABLE	–	–	–	–
POST_CONVOLUTION_COLOR_TABLE	–	–	–	–
POST_COLOR_MATRIX_COLOR_TABLE	–	–	–	–
COLOR_TABLE_FORMAT	–	–	–	–
COLOR_TABLE_WIDTH	–	–	–	–
COLOR_TABLE_RED_SIZE	–	–	–	–
COLOR_TABLE_GREEN_SIZE	–	–	–	–
COLOR_TABLE_BLUE_SIZE	–	–	–	–
COLOR_TABLE_ALPHA_SIZE	–	–	–	–
COLOR_TABLE_LUMINANCE_SIZE	–	–	–	–
COLOR_TABLE_INTENSITY_SIZE	–	–	–	–
COLOR_TABLE_SCALE	–	–	–	–
COLOR_TABLE_BIAS	–	–	–	–

Table 6.18: Pixels (cont.)

State	Exposed	Queriable	Common Get	Common-Lite Get
CONVOLUTION_1D	–	–	–	–
CONVOLUTION_2D	–	–	–	–
SEPARABLE_2D	–	–	–	–
CONVOLUTION	–	–	–	–
CONVOLUTION_BORDER_COLOR	–	–	–	–
CONVOLUTION_BORDER_MODE	–	–	–	–
CONVOLUTION_FILTER_SCALE	–	–	–	–
CONVOLUTION_FILTER_BIAS	–	–	–	–
CONVOLUTION_FORMAT	–	–	–	–
CONVOLUTION_WIDTH	–	–	–	–
CONVOLUTION_HEIGHT	–	–	–	–

Table 6.19: Pixels (cont.)

State	Exposed	Queriable	Common Get	Common-Lite Get
POST_CONVOLUTION_RED_SCALE	–	–	–	–
POST_CONVOLUTION_GREEN_SCALE	–	–	–	–
POST_CONVOLUTION_BLUE_SCALE	–	–	–	–
POST_CONVOLUTION_ALPHA_SCALE	–	–	–	–
POST_CONVOLUTION_RED_BIAS	–	–	–	–
POST_CONVOLUTION_GREEN_BIAS	–	–	–	–
POST_CONVOLUTION_BLUE_BIAS	–	–	–	–
POST_CONVOLUTION_ALPHA_BIAS	–	–	–	–
POST_COLOR_MATRIX_RED_SCALE	–	–	–	–
POST_COLOR_MATRIX_GREEN_SCALE	–	–	–	–
POST_COLOR_MATRIX_BLUE_SCALE	–	–	–	–
POST_COLOR_MATRIX_ALPHA_SCALE	–	–	–	–
POST_COLOR_MATRIX_RED_BIAS	–	–	–	–
POST_COLOR_MATRIX_GREEN_BIAS	–	–	–	–
POST_COLOR_MATRIX_BLUE_BIAS	–	–	–	–
POST_COLOR_MATRIX_ALPHA_BIAS	–	–	–	–
HISTOGRAM	–	–	–	–
HISTOGRAM_WIDTH	–	–	–	–
HISTOGRAM_FORMAT	–	–	–	–
HISTOGRAM_RED_SIZE	–	–	–	–
HISTOGRAM_GREEN_SIZE	–	–	–	–
HISTOGRAM_BLUE_SIZE	–	–	–	–
HISTOGRAM_ALPHA_SIZE	–	–	–	–
HISTOGRAM_LUMINANCE_SIZE	–	–	–	–
HISTOGRAM_SINK	–	–	–	–

Table 6.20: Pixels (cont.)

State	Exposed	Queriable	Common Get	Common-Lite Get
MINMAX	–	–	–	–
MINMAX_FORMAT	–	–	–	–
MINMAX_SINK	–	–	–	–
ZOOM_X	–	–	–	–
ZOOM_Y	–	–	–	–
PIXEL_MAP_I_TO_I	–	–	–	–
PIXEL_MAP_S_TO_S	–	–	–	–
PIXEL_MAP_I_TO_{RGBA}	–	–	–	–
PIXEL_MAP_R_TO_R	–	–	–	–
PIXEL_MAP_G_TO_G	–	–	–	–
PIXEL_MAP_B_TO_B	–	–	–	–
PIXEL_MAP_A_TO_A	–	–	–	–
PIXEL_MAP_*x*_TO_*y*_SIZE	–	–	–	–
READ_BUFFER	–	–	–	–

Table 6.21: Pixels (cont.)

State	Exposed	Queriable	Common Get	Common-Lite Get
ORDER	–	–	–	–
COEFF	–	–	–	–
DOMAIN	–	–	–	–
MAP1_*x*	–	–	–	–
MAP2_*x*	–	–	–	–
MAP1_GRID_DOMAIN	–	–	–	–
MAP2_GRID_DOMAIN	–	–	–	–
MAP1_GRID_SEGMENTS	–	–	–	–
MAP2_GRID_SEGMENTS	–	–	–	–
AUTO_NORMAL	–	–	–	–

Table 6.22: Evaluators

State	Exposed	Queriable	Common Get	Common-Lite Get
PERSPECTIVE_CORRECTION_HINT	✓	✓	**GetIntegerv**	**GetIntegerv**
POINT_SMOOTH_HINT	✓	✓	**GetIntegerv**	**GetIntegerv**
LINE_SMOOTH_HINT	✓	✓	**GetIntegerv**	**GetIntegerv**
POLYGON_SMOOTH_HINT	–	–	–	–
FOG_HINT	✓	✓	**GetIntegerv**	**GetIntegerv**
GENERATE_MIPMAP_HINT	✓	✓	**GetIntegerv**	**GetIntegerv**
TEXTURE_COMPRESSION_HINT	–	–	–	–

Table 6.23: Hints

State	Exposed	Queriable	Common Get	Common-Lite Get
MAX_LIGHTS	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_CLIP_PLANES	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_COLOR_MATRIX_STACK_DEPTH	–	–	–	–
MAX_MODELVIEW_STACK_DEPTH	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_PROJECTION_STACK_DEPTH	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_TEXTURE_STACK_DEPTH	✓	✓	**GetIntegerv**	**GetIntegerv**
SUBPIXEL_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_3D_TEXTURE_SIZE	–	–	–	–
MAX_TEXTURE_SIZE	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_CUBE_MAP_TEXTURE_SIZE	–	–	–	–
MAX_PIXEL_MAP_TABLE	–	–	–	–
MAX_NAME_STACK_DEPTH	–	–	–	–
MAX_LIST_NESTING	–	–	–	–
MAX_EVAL_ORDER	–	–	–	–
MAX_VIEWPORT_DIMS	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 6.24: Implementation Dependent Values

State	Exposed	Queriable	Common Get	Common-Lite Get
MAX_ATTRIB_STACK_DEPTH	–	–	–	–
MAX_CLIENT_ATTRIB_STACK_DEPTH	–	–	–	–
Maximum size of a color table	–	–	–	–
Maximum size of the histogram table	–	–	–	–
AUX_BUFFERS	–	–	–	–
RGBA_MODE	–	–	–	–
INDEX_MODE	–	–	–	–
DOUBLEBUFFER	–	–	–	–
ALIASED_POINT_SIZE_RANGE	✓	✓	**GetFloatv**	**GetFixedv**
SMOOTH_POINT_SIZE_RANGE	✓	✓	**GetFloatv**	**GetFixedv**
SMOOTH_POINT_SIZE_GRANULARITY	–	–	–	–
ALIASED_LINE_WIDTH_RANGE	✓	✓	**GetFloatv**	**GetFixedv**
SMOOTH_LINE_WIDTH_RANGE	✓	✓	**GetFloatv**	**GetFixedv**
SMOOTH_LINE_WIDTH_GRANULARITY	–	–	–	–

Table 6.25: Implementation Dependent Values (cont.)

State	Exposed	Queriable	Common Get	Common-Lite Get
MAX_CONVOLUTION_WIDTH	–	–	–	–
MAX_CONVOLUTION_HEIGHT	–	–	–	–
MAX_ELEMENTS_INDICES	–	–	–	–
MAX_ELEMENTS_VERTICES	–	–	–	–
MAX_TEXTURE_UNITS	✓	✓	**GetIntegerv**	**GetIntegerv**
SAMPLE_BUFFERS	✓	✓	**GetIntegerv**	**GetIntegerv**
SAMPLES	✓	✓	**GetIntegerv**	**GetIntegerv**
COMPRESSED_TEXTURE_FORMATS	✓	✓	**GetIntegerv**	**GetIntegerv**
NUM_COMPRESSED_TEXTURE_FORMATS	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 6.26: Implementation Dependent Values (cont.)

State	Exposed	Queriable	Common Get	Common-Lite Get
RED_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
GREEN_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
BLUE_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
ALPHA_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
INDEX_BITS	–	–	–	–
DEPTH_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
STENCIL_BITS	✓	✓	**GetIntegerv**	**GetIntegerv**
ACCUM_BITS	–	–	–	–

Table 6.27: Implementation Dependent Pixel Depths

State	Exposed	Queriable	Common Get	Common-Lite Get
LIST_BASE	–	–	–	–
LIST_INDEX	–	–	–	–
LIST_MODE	–	–	–	–
Server attribute stack	–	–	–	–
ATTRIB_STACK_DEPTH	–	–	–	–
Client attribute stack	–	–	–	–
CLIENT_ATTRIB_STACK_DEPTH	–	–	–	–
NAME_STACK_DEPTH	–	–	–	–
RENDER_MODE	–	–	–	–
SELECTION_BUFFER_POINTER	–	–	–	–
SELECTION_BUFFER_SIZE	–	–	–	–
FEEDBACK_BUFFER_POINTER	–	–	–	–
FEEDBACK_BUFFER_SIZE	–	–	–	–
FEEDBACK_BUFFER_TYPE	–	–	–	–
Current error code(s)	✓	✓	**GetError**	**GetError**
Corresponding error flags	✓	✓	–	–

Table 6.28: Miscellaneous

State	Exposed	Queriable	Common Get	Common-Lite Get
IMPLEMENTATION_COLOR_READ_TYPE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
IMPLEMENTATION_COLOR_READ_FORMAT_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
MATRIX_PALETTE_OES	✓	✓	**IsEnabled**	**IsEnabled**
MAX_PALETTE_MATRICES_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
MAX_VERTEX_UNITS_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
MATRIX_INDEX_ARRAY_OES	✓	✓	**IsEnabled**	**IsEnabled**
MATRIX_INDEX_ARRAY_SIZE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
MATRIX_INDEX_ARRAY_TYPE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
MATRIX_INDEX_ARRAY_STRIDE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
MATRIX_INDEX_ARRAY_POINTER_OES	✓	✓	**GetPointerv**	**GetPointerv**
MATRIX_INDEX_ARRAY_BUFFER_BINDING_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
WEIGHT_ARRAY_OES	✓	✓	**IsEnabled**	**IsEnabled**
WEIGHT_ARRAY_SIZE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
WEIGHT_ARRAY_TYPE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
WEIGHT_ARRAY_STRIDE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
WEIGHT_ARRAY_POINTER_OES	✓	✓	**GetPointerv**	**GetPointerv**
WEIGHT_ARRAY_BUFFER_BINDING_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
CURRENT_PALETTE_MATRIX_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
POINT_SPRITE_OES	✓	✓	**IsEnabled**	**IsEnabled**
COORD_REPLACE_OES	✓	✓	**GetTexEnviv**	**GetTexEnviv**
POINT_SIZE_ARRAY_OES	✓	✓	**IsEnabled**	**IsEnabled**
POINT_SIZE_ARRAY_TYPE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
POINT_SIZE_ARRAY_STRIDE_OES	✓	✓	**GetIntegerv**	**GetIntegerv**
POINT_SIZE_ARRAY_POINTER_OES	✓	✓	**GetPointerv**	**GetPointerv**
POINT_SIZE_ARRAY_BUFFER_BINDING_OES	✓	✓	**GetIntegerv**	**GetIntegerv**

Table 6.29: Core Additions and Extensions

Chapter 7

Core Additions and Extensions

An OpenGL ES profile consists of two parts: a subset of the full OpenGL pipeline, and some extended functionality that is drawn from a set of OpenGL ES-specific extensions to the full OpenGL specification. Each extension is pruned to match the profile's command subset and added to the profile as either a core addition or a profile extension. Core additions differ from profile extensions in that the commands and tokens do not include extension suffixes in their names.

Profile extensions are further divided into required (mandatory) and optional extensions. Required extensions must be implemented as part of a conforming implementation, whereas the implementation of optional extensions is left to the discretion of the implementor. Both types of extensions use extension suffixes as part of their names, are present in the `EXTENSIONS` string, and participate in function address queries defined in the platform embedding layer. Required extensions have the additional packaging constraint, that commands defined as part of a required extension must also be available as part of a static binding if core commands are also available in a static binding. The commands comprising an optional extension may optionally be included as part of a static binding.

From an API perspective, commands and tokens comprising a core addition are indistinguishable from the original OpenGL subset. However, to increase application portability, an implementation may also implement a core addition as an extension by including suffixed versions of commands and tokens in the appropriate dynamic and optional static bindings and the extension name in the `EXTENSIONS` string.

> ■ Extensions preserve all traditional extension properties regardless of whether they are required or optional. Required extensions must be present; therefore, additionally providing static bindings simplifies application usage and reinforces the ubiquity of the extension. Permitting core additions to be included as extensions allows extensions that are promoted to core additions in later profile revisions to continue to be available as extensions, retaining application compatibility. ❑

The Common and Common-Lite profiles add subsets of the `OES_byte_coordinates`, `OES_fixed_point`, `OES_single_precision` and `OES_matrix_get` ES-specific extensions as core additions; `OES_read_format`, `OES_compressed_paletted_texture`, `OES_point_size_array` and `OES_point_sprite` as required profile extensions; and `OES_matrix_palette` and `OES_draw_texture` as optional profile extensions.

The `OES_query_matrix` optional extension in OpenGL ES 1.0 has been deprecated in OpenGL ES 1.1. The various matrices in GL can be obtained by calling the fixed, float version of get or by using the `OES_matrix_get` core extension.

Extension Name	Common	Common-Lite
OES_byte_coordinates	core addition	core addition
OES_fixed_point	core addition	core addition
OES_single_precision	core addition	n/a
OES_matrix_get	core addition	core addition
OES_read_format	required extension	required extension
OES_compressed_paletted_texture	required extension	required extension
OES_point_size_array	required extension	required extension
OES_point_sprite	required extension	required extension
OES_matrix_palette	optional extension	optional extension
OES_draw_texture	optional extension	optional extension

Table 7.1: OES Extension Disposition

7.1 Byte Coordinates

The OES_byte_coordinates extension allows byte data types to be used as vertex and texture coordinates. The Common/Common-Lite profile supports byte coordinates in vertex array commands.

7.2 Fixed Point

The OES_fixed_point extension defines an integer fixed-point data type for vertex attributes and command parameters. The extension specification includes commands that parallel all OpenGL 1.5 commands with floating-point parameters (including commands that support a single parameter type version such as **DepthRange**, **PointSize**, and **LineWidth**). The subset of commands included in the Common and Common-Lite profiles matches exactly the subset of floating-point commands included in the Common profile. The subset of commands is summarized in Table 7.2

Normal3x(fixed coords)
MultiTexCoord4x(fixed coords)
Color4x(fixed coords)
VertexPointer(int size, enum type, sizei stride, const void *ptr) size = 2,3,4 type = FIXED
ColorPointer(int size, enum type, sizei stride, const void *ptr) size=3,4 type = FIXED
NormalPointer(enum type, sizei stride, const void *ptr) type = FIXED
TexCoordPointer(int size, enum type, sizei stride, const void *ptr) size = 2,3,4 type = FIXED
DepthRangex(clampx n, clampx f)
LoadMatrixx(fixed m[16])
MultMatrixx(fixed m[16])
Rotatex(fixed angle, fixed x, fixed y, fixed z)

Scalex(fixed x, fixed y, fixed z)
Translatex(fixed x, fixed y, fixed z)
Frustumx(fixed l, fixed r, fixed b, fixed t, fixed n, fixed f)
Orthox(fixed l, fixed r, fixed b, fixed t, fixed n, fixed f)
ClipPlanex(enum plane, const fixed *equation)
Materialx[v](enum face, enum pname, T param)
Lightx[v](enum light, enum pname, T param)
LightModelx[v](enum pname, T param)
PointParameterx[v](enum pname, T param)
PointSizex(fixed size)
LineWidthx(fixed width)
PolygonOffsetx(fixed factor, fixed units)
TexParameterx(enum target, enum pname, T param)
TexEnvx[v](enum target, enum pname, T param)
Fogx[v](enum pname, T param)
SampleCoveragex(clampx value, boolean invert)
AlphaFuncx(enum func, clampx ref)
ClearColorx(clampx red, clampx green, clampx blue, clampx alpha)
ClearDepthx(clampx depth)
GetFixedxv(enum pname, T *params)
GetClipPlanex(enum pname, T eqn[4])
GetLightxv(enum light, enum pname, T *params)
GetMaterialxv(enum face, enum pname, T *params)
GetTexEnvxv(enum env, enum pname, T *params)
GetTexParameterxv(enum target, enum pname, T *params)

Table 7.2: Common/Common-Lite profile subset of OES_fixed_point

7.3 Single-precision Commands

The OES_single_precision_commands extension creates new single-precision parameter command variants of commands that have no such variants (**DepthRange**, **TexGen**, **Frustum**, **Ortho**, etc.). Only the subset matching the profile feature set is included in the Common profile.

DepthRangef(clampf n, clampf f)
Frustumf(float l, float r, float b, float t, float n, float f)
Orthof(float l, float r, float b, float t, float n, float f)
ClearDepthf(clampf depth)

GetClipPlanef(enum pname, float eqn[4])

7.4 Compressed Paletted Texture

The OES_compressed_paletted_texture extension provides a method for specifying a compressed texture image as a color index image accompanied by a palette. The extension adds ten new texture internal formats to specify different combinations of index width and palette color format: PALETTE4_RGB8_OES, PALETTE4_RGBA8_OES, PALETTE4_R5_G6_B5_OES, PALETTE4_RGBA4_OES, PALETTE4_RGB5_A1_OES, PALETTE8_RGB8_OES, PALETTE8_RGBA8_OES, PALETTE8_R5_G6_B5_OES, PALETTE8_RGBA4_OES, and PALETTE8_RGB5_A1_OES. The state queries for NUM_COMPRESSED_TEXTURE_FORMATS and COMPRESSED_TEXTURE_FORMATS include these formats.

7.5 Read Format

The OES_read_format extension allows implementation-specific pixel type and format parameters to be queried by an application and used in **ReadPixel** commands. The format and type values must be from the set of supported texture image format and type values specified in Table 3.1.

7.6 Matrix Palette

The optional OES_matrix_palette extension adds the ability to support vertex skinning in OpenGL ES. This extension allow OpenGL ES to support a palette of matrices. The matrix palette defines a set of matrices that can be used to transform a vertex. The matrix palette is not part of the model view matrix stack and is enabled by setting the MATRIX_MODE to MATRIX_PALETTE_OES.

The n vertex units use a palette of m modelview matrices (where n and m are constrained to implementation defined maxima). Each vertex has a set of n indices into the palette, and a corresponding set of n weights. Matrix indices and weights can be changed for each vertex.

When this extension is utilized, the enabled units transform each vertex by the modelview matrices specified by the vertices' respective indices. These results are subsequently scaled by the weights of the respective units and then summed to create the eyespace vertex.

7.7 Point Sprites

The OES_point_sprites extension provides a method for application to draw particles using points instead of quads. This extension also allows an app to specify texture coordinates that are interpolated across the point instead of the same texture coordinate used by traditional GL points.

7.8 Point Size Array

This OES_point_size_array extension extends how points and point sprites are rendered by allowing an array of point sizes instead of a fixed input point size given by PointSize. This provides flexibility for applications to do particle effects.

The vertex arrays will be extended to include a point size array. The point size array can be enabled/disabled via POINT_SIZE_ARRAY_OES. The point size array, if enabled, controls the sizes used to render points and point sprites. If point size array is enabled, the point size defined by PointSize is ignored. The point sizes supplied in the point size arrays will be the sizes used to render both points and point sprites.

Distance-based attenuation works in conjunction with POINT_SIZE_ARRAY_OES. If distance-based attenuation is enabled the point size from the point size array will be attenuated as defined by point parameters to compute the final point size.

7.9 Matrix Get

Many applications require the ability to be able to read the GL matrices. OpenGL ES 1.1 will allow an application to read the matrices using the GetFloatv command for the common profile and the GetFixedv command for the common-lite profile.

In cases where the common-lite implementation stores matrices and performs matrix operations internally using floating point (example would be OpenGL ES implementations that support JSR184 etc.) the GL cannot return the floating pt matrix elements since the float data type is not supported by the common-lite profile. Using GetFixedv to get the matrix data will result in a loss of information.

To take care of this issue, new tokens are proposed by this extension. These tokens will allow the GL to return a representation of the floating pt matrix elements as as an array of integers, according to the IEEE 754 floating pt "single format" bit layout.

7.10 Draw Texture

This `OES_draw_texture` extension defines a mechanism for writing pixel rectangles from one or more textures to a rectangular region of the screen. This capability is useful for fast rendering of background paintings, bitmapped font glyphs, and 2D framing elements in games

Chapter 8

Packaging

8.1 Header Files

The header file structure is the same as in a full OpenGL distribution, using a single header file: `gl.h`. Additional enumerants `VERSION_ES_CM_x_y` and `VERSION_ES_CL_x_y`, where `x` and `y` are the major and minor version numbers as described in Section 6.1, are included in the header file. These enumerants indicate the versions of profiles supported at compile-time.

8.2 Libraries

Each profile defines a distinct link-library. The library name includes the profile name as `libGLES_nn.z` where `nn` is either `CM` or `CL` and `.z` is a platform-specific library suffix (i.e., `.a`, `.so`, `.lib`, etc.). The symbols for the platform-specific embedding library are also included in the link-library. By default, the library name `libGLES_nn.z` includes EGL entry points. The OpenGL ES library that does not include EGL entry points will be named `libGLESv1_nn.z`. Availability of static and dynamic function bindings is platform dependent. Rules regarding the export of bindings for core additions, required profile extensions, and optional platform extensions are described in Chapter 7.

Appendix A

Acknowledgements

The OpenGL ES Common and Common-Lite profiles are the result of the contributions of many people, representing a cross section of the desktop, hand-held, and embedded computer industry. Following is a partial list of the contributors, including the company that they represented at the time of their contribution:

Aaftab Munshi, ATI

Andy Methley, Panasonic

Axel Mamode, Sony Computer Entertainment

Barthold Lichtenbelt, 3Dlabs

Benji Bowman, Imagination Technologies

Borgar Ljosland, Falanx

Brian Murray, Motorola

Bryce Johnstone, Texas Instruments

Carlos Sarria, Imagination Technologies

Chris Tremblay, Motorola

Claude Knaus, Esmertec

Clay Montgomery, Nokia

Dan Petersen, Sun

Dan Rice, Sun

David Blythe, HI

David Yoder, Motorola

Doug Twilleager, Sun

Ed Plowman, ARM

Graham Connor, Imagination Technologies

Greg Stoner, Motorola

Hannu Napari, Hybrid

Harri Holopainen, Hybrid

Jacob Ström, Ericsson

Jani Vaarala, Nokia
Jerry Evans, Sun
John Metcalfe, Imagination Technologies
Jon Leech, Silicon Graphics
Kari Pulli, Nokia
Lane Roberts, Symbian
Madhukar Budagavi, Texas Instruments
Mathias Agopian, PalmSource
Mark Callow, HI
Mark Tarlton, Motorola
Mike Olivarez, Motorola
Neil Trevett, 3Dlabs
Nick Triantos, Nvidia
Petri Kero, Hybrid
Petri Nordlund, Bitboys
Phil Huxley, Tao Group
Remi Arnaud, Sony Computer Entertainment
Robert Simpson, Bitboys
Tero Sarkinnen, Futuremark
Timo Suoranta, Futuremark
Thomas Tannert, Silicon Graphics
Tomi Aarnio, Nokia
Tom McReynolds, Nvidia
Tom Olson, Texas Instruments
Ville Miettinen, Hybrid Graphics

Appendix B

OES Extension Specifications

B.1 OES_byte_coordinates

B.2 OES_fixed_point

B.3 OES_single_precision

B.4 OES_read_format

B.5 OES_query_matrix

B.6 OES_compressed_paletted_texture

B.7 OES_matrix_palette

B.8 OES_point_sprite

B.9 OES_point_size_array

B.10 OES_matrix_get

B.11 OES_draw_texture