OpenGL® ES
Common/Common-Lite Profile Specification

Version 1.1.12 (Difference Specification) (Annotated)

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

April 24, 2008

Copyright (c) 2002-2008 The Khronos Group Inc. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be re-formatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group web-site should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

Khronos is a trademark of The Khronos Group Inc. OpenGL is a registered trademark, and OpenGL ES is a trademark, of Silicon Graphics, Inc.

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 26

4 Per-Fragment Operations and the Framebuffer 27
4.1 Per-Fragment Operations 27
4.1.1 Blending 29
4.2 Whole Framebuffer Operations 30
4.3 Drawing, Reading, and Copying Pixels 30

5 Special Functions 32
5.1 Evaluators 32
5.2 Selection 32
5.3 Feedback 33
5.4 Display Lists 33
5.5 Flush and Finish 33
5.6 Hints 34

6 State and State Requests 35
6.1 Querying GL State 35
6.2 State Tables 37

7 Core Additions and Extensions 54
7.1 Byte Coordinates 55
7.2 Fixed Point 55
7.3 Single-precision Commands 56
7.4 Compressed Paletted Texture 57
7.5 Read Format 57
7.6 Matrix Palette 57
7.7 Point Sprites 57
7.8 Point Size Array 57
7.9 Matrix Get 58
7.10 Draw Texture 58

8 Packaging 59

A Acknowledgements 60

B OES Extension Specifications 62
B.1 OES_byte_coordinates 62
B.2 OES_fixed_point 65
B.3 OES_single_precision 74
B.4 OES_read_format 79
B.5 OES_query_matrix 83
B.6 OES_compressed_paletted_texture 86
B.7 OES_matrix_palette 92
B.8 OES_point_sprite 100
B.9 OES_point_size_array 107
B.10 OES_matrix_get 110
B.11 OES_draw_texture 113

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 $\ast$ 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 internal format.
- 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.

The definition of texture functions REPLACE, MODULATE, and DECAL in table 3.22 of the OpenGL 1.5 specification, and of texture functions BLEND and ADD in table 3.23 of the OpenGL 1.5 specification, are altered by replacing the term C_f with C_p everywhere it occurs. The resulting texture functions are as defined in tables 3.5 and 3.6 below.

The reason for this change is not to alter behavior relative to desktop OpenGL 1.5 implementations, but to correct an error in the OpenGL 1.5 specification. Actual implementations of OpenGL 1.5 perform as described below. The same correction was made in the OpenGL 2.0 specification, but a corrected version of the OpenGL 1.5 specification was never issued.

Texture Base Internal Format	`REPLACE` Function	`MODULATE` Function	`DECAL` Function
`ALPHA`	$C_v = C_p$ $A_v = A_s$	$C_v = C_p$ $A_v = A_pA_s$	*undefined*
`LUMINANCE` (or 1)	$C_v = C_s$ $A_v = A_p$	$C_v = C_pC_s$ $A_v = A_p$	*undefined*
`LUMINANCE_ALPHA` (or 2)	$C_v = C_s$ $A_v = A_s$	$C_v = C_pC_s$ $A_v = A_pA_s$	*undefined*
`RGB` (or 3)	$C_v = C_s$ $A_v = A_p$	$C_v = C_pC_s$ $A_v = A_p$	$C_v = C_s$ $A_v = A_p$
`RGBA` (or 4)	$C_v = C_s$ $A_v = A_s$	$C_v = C_pC_s$ $A_v = A_pA_s$	$C_v = C_p(1 - A_s) + C_sA_s$ $A_v = A_p$

Table 3.5: Texture functions `REPLACE`, `MODULATE`, and `DECAL`. Replaces table 3.22 of the OpenGL 1.5 specification.

Texture Base Internal Format	`BLEND` Function	`ADD` Function
`ALPHA`	$C_v = C_p$ $A_v = A_pA_s$	$C_v = C_p$ $A_v = A_pA_s$
`LUMINANCE` (or 1)	$C_v = C_p(1 - C_s) + C_cC_s$ $A_v = A_p$	$C_v = C_p + C_s$ $A_v = A_p$
`LUMINANCE_ALPHA` (or 2)	$C_v = C_p(1 - C_s) + C_cC_s$ $A_v = A_pA_s$	$C_v = C_p + C_s$ $A_v = A_pA_s$
`RGB` (or 3)	$C_v = C_p(1 - C_s) + C_cC_s$ $A_v = A_p$	$C_v = C_p + C_s$ $A_v = A_p$
`RGBA` (or 4)	$C_v = C_p(1 - C_s) + C_cC_s$ $A_v = A_pA_s$	$C_v = C_p + C_s$ $A_v = A_pA_s$

Table 3.6: Texture functions `BLEND` and `ADD`. Replaces table 3.23 of the OpenGL 1.5 specification.

`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`.

SRCn_RGB	OPERANDn_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.

SRCn_ALPHA	OPERANDn_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

Appendix C.4 of the Full Specification, and the Khronos API Implementers Guide referred to from that appendix, describe recommended and required practice for implementing OpenGL ES, including names of header files and libraries making up the implementation, and links to standard versions of the header files defining interfaces for the core OpenGL ES API

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

```
Name

    OES_byte_coordinates

Name Strings

    GL_OES_byte_coordinates

Contact

    Kari Pulli, Nokia (kari.pulli ’at’ nokia.com)

Status

    Ratified by the Khronos BOP, July 23, 2003.

Version

    $Date: 2003/07/23 04:23:25 $ $Revision: 1.5 $

Number

    OpenGL ES Extension #4 (formerly OpenGL Extension #291)

Dependencies

    OpenGL 1.1 is required.

Overview

    This extension allows specifying, additionally to all existing
    values, byte-valued vertex and texture coordinates to be used.

    The main reason for introducing the byte-argument is to allow
    storing data more compactly on memory-restricted environments.
```

IP Status

There is no intellectual property associated with this extension.

Issues

None known.

New Procedures and Functions

None

New Tokens

Accepted by the <type> parameter of VertexPointer and TexCoordPointer

```
BYTE     0x1400
```

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

Add signed byte entry points to first paragraph of section 2.7 (Vertex Specification):

```
void Vertex{234}bOES( T coords );
void Vertex{234}bvOES( T coords );
```

and to the second paragraph:

```
void TexCoord{1234}bOES( T coords );
void TexCoord{1234}bvOES( T coords );
```

and to the third paragraph:

```
void MultiTexCoord{1234}bOES( enum texture, T coords );
void MultiTexCoord{1234}bvOES( enum texture, T coords );
```

Add byte to supported types in Table 2.4 (Vertex Array Sizes):

Command	Sizes	Types
VertexPointer	2,3,4	byte,short,int,float,double
TexCoordPointer	1,2,3,4	byte,short,int,float,double

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

None

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

None

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

None

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

None

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance)

None

Additions to the AGL/GLX/WGL Specifications

GLX Protocol

Byte type commands are mapped on the client-side to the appropriate short or int command protocol.

Errors

No new errors, giving byte as <type> argument to VertexPointer or TexCoordPointer is not an error any more.

New State

(table 6.6, pp. 214-215)

Get Value	Type	Get Command	Value	Description	Sec.	Attribute
VERTEX_x	Z_5	GetIntegerv	FLOAT	Type of vertex coordinates	2.8	vertex-array
TEXTURE_COORD_x	2 * x Z_5	GetIntegerv	FLOAT	Type of texture coordinates	2.8	vertex-array

_x = _ARRAY_TYPE

New Implementation Dependent State

None

Revision History

Sep 23, 2002	Kari Pulli	Created the document
Sep 26, 2002	Kari Pulli	Incorporated comments by Jon Leech
Feb 26, 2003	David Blythe	Changed prefix to OES
Jul 08, 2003	David Blythe	Deleted Dependencies on section, added extension number, narrow state table
Jul 11, 2003	David Blythe	Changed to use OES suffixes
Jul 12, 2003	David Blythe	Added note about GLX protocol

B.2 OES_fixed_point

Name

OES_fixed_point

Name Strings

GL_OES_fixed_point

Contact

David Blythe (blythe ’at’ bluevoid.com)

Status

Ratified by the Khronos BOP, July 23, 2003.
Ratified by the Khronos BOP, Aug 5, 2004.

Version

Last Modifed Date: 16 June 2004
Author Revision: 0.8

Number

OpenGL ES Extension #9 (formerly ARB Extension #292)

Dependencies

None
The extension is written against the OpenGL 1.3 Specification.

Overview

This extension provides the capability, for platforms that do not have efficient floating-point support, to input data in a fixed-point format, i.e., a scaled-integer format. There are several ways a platform could try to solve the problem, such as using integer only commands, but there are many OpenGL commands that have only floating-point or double-precision floating-point parameters. Also, it is likely that any credible application running on such a platform will need to perform some computations and will already be using some form of fixed-point representation. This extension solves the problem by adding new ‘‘fixed’, and ‘‘clamp fixed’’ data types based on a a two’s complement S15.16 representation. New versions of commands are created with an ’x’ suffix that take fixed or clampx parameters.

IP Status

None

Issues

* Add double-precision (S31.32) form too?
NO

* Additional InterleavedArray formats?
NO

* Should newly suffixed commands, e.g., PointSize, get an alias with a float or double suffix for consistency?
NO

* Are enums converted to fixed by scaling by 2^16.
NO. An enums are passed through as if they are already in S15.16 form. Requiring scaling is too error prone.

New Procedures and Functions

NOTE: ‘T’ expands to ’const fixed*’ or ‘fixed’ as appropriate

```
void Vertex{234}x[v]OES(T coords);
void Normal3x[v]OES(T coords);
void TexCoord{1234}x[v]OES(T coords);
void MultiTexCoord{1234}x[v]OES(enum texture, T coords);
void Color{34}x[v]OES(T components);
void Indexx[v]OES(T component);
void RectxOES(fixed x1, fixed y1, fixed x2, fixed y2);
void RectxvOES(const fixed v1[2], const fixed v2[2]);

void DepthRangexOES(clampx n, clampx f);
void LoadMatrixxOES(const fixed m[16]);
void MultMatrixxOES(const fixed m[16]);
void LoadTransposeMatrixxOES(const fixed m[16]);
void MultTransposeMatrixxOES(const fixed m[16]);
void RotatexOES(fixed angle, fixed x, fixed y, fixed z);
void ScalexOES(fixed x, fixed y, fixed z);
void TranslatexOES(fixed x, fixed y, fixed z);
void FrustumxOES(fixed l, fixed r, fixed b, fixed t, fixed n, fixed f);
void OrthoxOES(fixed l, fixed r, fixed b, fixed t, fixed n, fixed f);
void TexGenx[v]OES(enum coord, enum pname, T param);
void GetTexGenxvOES(enum coord, enum pname, T* params);

void ClipPlanexOES(enum plane, const fixed* equation);
void GetClipPlanexOES(enum plane, fixed* equation);

void RasterPos{234}x[v]OES(T coords);

void Materialx[v]OES(enum face, enum pname, T param);
void GetMaterialxOES(enum face, enum pname, T param);
void Lightx[v]OES(enum light, enum pname, T* params);
void GetLightxOES(enum light, enum pname, T* params);
void LightModelx[v]OES(enum pname, T param);

void PointSizexOES(fixed size);
void PointParameterxvOES(enum pname, const fixed *params)
void LineWidthxOES(fixed width);
void PolygonOffsetxOES(fixed factor, fixed units);

void PixelStorex{enum pname, T param);
void PixelTransferxOES(enum pname, T param);
void PixelMapx{enum map int size T* values);
void GetPixelMapxv{enum map int size T* values);

void ConvolutionParameterx[v]OES(enum target, enum pname, T param);
void GetConvolutionParameterxvOES(enum target, enum pname, T* params);
void GetHistogramParameterxvOES(enum target, enum pname, T *params);

void PixelZoomxOES(fixed xfactor, fixed yfactor);

void BitmapxOES(sizei width, sizei height, fixed xorig, fixed yorig,
             fixed xmove, fixed ymove, const ubyte* bitmap);

void TexParameterx[v]OES(enum target, enum pname, T param);
void GetTexParameterxvOES(enum target, enum pname, T* params);
void GetTexLevelParameterxvOES(enum target, int level, enum pname, T* params);
void PrioritizeTexturesxOES(sizei n, uint* textures, clampx* priorities);
void TexEnvx[v]OES(enum target, enum pname, T param);
void GetTexEnvxvOES(enum target, enum pname, T* params);

void Fogx[v]OES(enum pname, T param);

void SampleCoverageOES(clampx value, boolean invert);
void AlphaFuncxOES(enum func, clampx ref);

void BlendColorxOES(clampx red, clampx green, clampx blue, clampx alpha);

void ClearColorxOES(clampx red, clampx green, clampx blue, clampx alpha);
void ClearDepthxOES(clampx depth);
void ClearAccumxOES(clampx red, clampx green, clampx blue, clampx alpha);
void AccumxOES(enum op, fixed value);

void Map1xOES(enum target, T u1, T u2, int stride, int order, T points);
void Map2xOES(enum target, T u1, T u2, int ustride, int uorder,
                       T v1, T v2, int vstride, int vorder, T points);
void MapGrid1xOES(int n, T u1, T u2);
void MapGrid2xOES(int n, T u1, T u2, T v1, T v2);
void GetMapxvOES(enum target, enum query, T* v);
void EvalCoord{12}x[v]OES(T coord);

void FeedbackBufferxOES(sizei n, enum type, fixed* buffer);
void PassThroughxOES(fixed token);
```

```
    GetFixedvOES(enum pname, fixed* params);
```

New Tokens

```
    FIXED_OES                0x140C
```

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

Section 2.1.1 Floating-Point Computation

Add the following paragraphs:

On some platforms, floating-point computations are not sufficiently well supported to be used in an OpenGL implementation. On such platforms, fixed-point representations may be a viable substitute for floating-point. Internal computations can use either fixed-point or floating-point arithmetic. Fixed-point computations must be accurate to within +/-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*0 = 0*x = 0. 1*x = x*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.

Section 2.3 GL Command Syntax

Paragraph 3 is updated to include the 'x' suffix and

Table 2.1 is modified to include the row:

x	fixed

Table 2.2 is modified to include the rows:

fixed	32	signed 2's complement S15.16 scaled integer
clampx	32	S15.16 scaled integer clamped to [0, 1]

and the count of the number of rows in the text is changed to 16.

Add paragraph

The mapping of GL data types to data types of a specific language binding are part of the language binding definition and may be platform-dependent. Type conversion and type promotion behavior when mixing actual and formal arguments of different data types are specific to the language binding and platform. For example, the C language includes automatic conversion between integer and floating-point data types, but does not include automatic conversion between the int and fixed or float and fixed GL types since the fixed data type is not a distinct built-in type. Regardless of language binding, the enum type converts to fixed-point without scaling and integer types are converted by multiplying by 2^16.

Section 2.7 Vertex Specification

Commands are revised to included ’x’ suffix.

Section 2.8 Vertex Arrays

Table 2.4 Vertex Array Sizes is revised to include the ’fixed’ type for all commands except EdgeFlagPointer.

References to Vertex command suffixes are revised to include ’x’.

Section 2.9 Rectangles

Revise to include ’x’ suffix.

Section 2.10 Coordinate Transformations

Revise to include ’x’ suffix. Section 2.10.1 describes clampx. Add alternate suffixed versions of Ortho and Frustum.

Section 2.11 Clipping

Add alternate suffixed version of ClipPlane.

Section 2.12 Current Raster Position

Revise to include ’x’ suffix.

Section 2.13 Colors and Coloring

Revise to include ’x’ suffix and
Table 2.6 is modified to include row:

fixed	c

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

Section 3.3 Points

Add alternate suffixed PointSize command.

Section 3.4 Line Segments

Add alternate suffixed LineWidth command.

Section 3.5 Polygons

Add alternate suffixed PolygonOffset command.

Section 3.6 Pixel Rectangles

Revise to include ’x’ suffix on PixelStore, PixelTransfer, PixelMap, ConvolutionParameter.

Table 3.5 is modified to include row:

FIXED	fixed	No

Add alternate suffixed PixelZoom to Section 3.6.5

Section 3.7 Bitmaps

Add alternate suffixed Bitmap command.

Section 3.8 Texturing

Revise to include ’x’ suffix in TexParameter (Section 3.8.4).

Add alternate suffixed PrioritizeTextures command (Section 3.8.11).

Revise to include ’x’ suffix in TexEnv (Section 3.8.12).

Section 3.10 Fog

Revise to include ;x; suffix in Fog command.

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

Section 4.1 Fragment Operations

Add alternate suffixed SampleCoverage command (Section 4.1.3), AlphaFunc command (Section 4.1.4), BlendColor command (Section 4.1.7).

Section 4.2 Whole Framebuffer Operations

Add alternate suffixed ClearColor, ClearDepth, and ClearAccum commands (Section 4.2.3).

Add alternate suffixed Accum command (Section 4.2.4).

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

Section 5.1 Evaluators

Revise to include ’x’ suffix on Map1, Map2, Map1Grid, and Map2Grid commands.

Section 5.3 Feedback

Add alternate suffixed FeedbackBuffer and PassThrough commands.
Revise Figure 5.2 to indicate ’f’ values may also be ’x’ values.

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

Add GetFixedv to Section 6.1.1. Revise Section 6.1.2 to include implied conversions for GetFixedv.

Revise to include ’x’ suffix for GetClipPlane, GetLightm GetMaterial, GetTexEnv, GetTexGen, GetTexParameter, GetTexLevelParameter, GetPixelMap, and GetMap in Section 6.1.3.

Revise to include ’x’ suffix for GetHistogramParameter (Section 6.1.9).

Section 6.2 State Tables

Revise intro paragraph to include GetFixedv.

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance)

None

Additions to the AGL/GLX/WGL Specifications

None

Additions to the WGL Specification

None

Additions to the AGL Specification

None

Additions to Chapter 2 of the GLX 1.3 Specification (GLX Operation)

The data representation is client-side only. The GLX layer performs translation between fixed and float representations.

Additions to Chapter 3 of the GLX 1.3 Specification (Functions and Errors)

Additions to Chapter 4 of the GLX 1.3 Specification (Encoding on the X Byte Stream)

Additions to Chapter 5 of the GLX 1.3 Specification (Extending OpenGL)

Additions to Chapter 6 of the GLX 1.3 Specification (GLX Versions)

GLX Protocol

Fixed type entry points are mapped on the client-side to the appropriate floating-point command protocol. To preserve precision, double-precision protocol is encouraged, but not required.

Errors

None

New State

None

New Implementation Dependent State

None

Revision History

12/15/2002   0.1
- Original draft.

03/31/2003   0.2
- Corrected a typo in GetClipPlanex and FIXED_OES.

04/24/2003   0.3
- Added clarification that enums must be converted to fixed by scaling when passed in a fixed parameter type. Corrected some typos.

05/29/2003   0.4
- Changed enums to be passed unscaled when passed to a fixed formal parameter.

07/08/2003   0.5
- Removed bogus Dependencies on section
- Added extension number and enumerant value

```
07/11/2003    0.6
    - Added OES suffixes

07/12/2003    0.7
    - Added note about GLX protocol

06/16/2004    0.8
    - Added ClipPlanex, and various Get functions
```

B.3 OES_single_precision

Name

OES_single_precision

Name Strings

GL_OES_single_precision

Contact

David Blythe (blythe ’at’ bluevoid.com)

Status

Ratified by the Khronos BOP, July 23, 2003.
Ratified by the Khronos BOP, Aug 5, 2004.

Version

Last Modifed Date: April 22, 2007
Author Revision : 0.6

Number

OpenGL ES Extension #18 (formerly OpenGL Extension #293)

Dependencies

None
The extension is written against the OpenGL 1.3 Specification.

Overview

This extension adds commands with single-precision floating-point parameters corresponding to the commands that only variants that accept double-precision floating-point input. This allows an application to avoid using double-precision floating-point data types. New commands are added with an ’f’ prefix.

IP Status

None

Issues

- An alternative is to suggest platforms define GLfloat and GLdouble to be the same type, since it is unlikely that both single- and double-precision are required at the same time.

Resolved: This might create additional confusion, so it is better to define new commands.

New Procedures and Functions

```
void DepthRangefOES(clampf n, clampf f);
void FrustumfOES(float l, float r, float b, float t, float n, float f);
void OrthofOES(float l, float r, float b, float t, float n, float f);

void ClipPlanefOES(enum plane, const float* equation);
void GetClipPlanefOES(enum plane, float* equation);

void glClearDepthfOES(clampd depth);
```

New Tokens

None

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

Section 2.10 Coordinate Transformations

Revise to include ’f’ suffix.
Add alternate suffixed versions of DepthRange (2.10.1).
Add alternate suffixed versions of Ortho and Frustum (2.10.2).

Section 2.11 Clipping

Add alternate suffixed version of ClipPlane.

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

None

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

Section 4.2.3 Clearing the Buffers

Add alternate suffixed version of ClearDepth.

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

None

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

None

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance)

None

Additions to the AGL/GLX/WGL Specifications

None

Additions to the WGL Specification

None

Additions to the AGL Specification

None

Additions to Chapter 2 of the GLX 1.3 Specification (GLX Operation)

The data representation is client-side only. The GLX layer performs translation between float and double representations.

Additions to Chapter 3 of the GLX 1.3 Specification (Functions and Errors)

Additions to Chapter 4 of the GLX 1.3 Specification (Encoding on the X Byte Stream)

Additions to Chapter 5 of the GLX 1.3 Specification (Extending OpenGL)

Additions to Chapter 6 of the GLX 1.3 Specification (GLX Versions)

GLX Protocol

Five new GL rendering commands are added. The following commands are sent to the server as part of a glXRender request:

```
ClearDepthfOES
    2           8               rendering command length
    2           4308            rendering command opcode
    4           FLOAT32         z

DepthRangefOES
    2           12              rendering command length
    2           4309            rendering command opcode
    4           FLOAT32         n
    4           FLOAT32         f

FrustumfOES
    2           28              rendering command length
    2           4310            rendering command opcode
    4           FLOAT32         l
    4           FLOAT32         r
    4           FLOAT32         b
    4           FLOAT32         t
    4           FLOAT32         n
```

```
            4           FLOAT32         f

        OrthofOES
            2           28              rendering command length
            2           4311            rendering command opcode
            4           FLOAT32         l
            4           FLOAT32         r
            4           FLOAT32         b
            4           FLOAT32         t
            4           FLOAT32         n
            4           FLOAT32         f

        ClipPlanefOES
            2           24              rendering command length
            2           4312            rendering command opcode
            4           ENUM            plane
            4           FLOAT32         v[0]
            4           FLOAT32         v[1]
            4           FLOAT32         v[2]
            4           FLOAT32         v[3]


    The remaining commands are non-rendering commands.  These commands are
    sent separately (i.e., not as part of a glXRender or glXRenderLarge
    request), using the glXVendorPrivateWithReply request:

        GetClipPlanefOES
            1           CARD8           opcode (X assigned)
            1           17              GLX opcode (glXVendorPrivateWithReply)
            2           4               request length
            4           1421            vendor specific opcode
            4           GLX_CONTEXT_TAG context tag
            4           ENUM            plane
          =>
            1           1               reply
            1                           unused
            2           CARD16          sequence number
            4           0               reply length
            4           FLOAT32         v[0]
            4           FLOAT32         v[1]
            4           FLOAT32         v[2]
            4           FLOAT32         v[3]
            8                           unused

Errors

    None

New State

    None
```

New Implementation Dependent State

None

Revision History

03/27/2003 0.1
- First draft created.

07/08/2003 0.2
- Delete unused Dependencies on section
- Added extension number

07/09/2003 0.3
- Added missing ClearDepthfOES
- Removed ’_’s from names.

07/22/2003 0.4
- Added GLX protocol (Thomas Roell)

06/28/2004 0.5
- Added ClipPlanef function (Aaftab Munshi)

04/22/2007 0.6
- Fixed ’void void’ typo (Jon Leech)

B.4 OES_read_format

Name

OES_read_format

Name Strings

GL_OES_read_format

Contact

Aaftab Munshi (amunshi@ati.com)

Status

Revision 0.2 ratified by the Khronos BOP, July 23, 2003.
Revision 0.3 to be ratified

Version

Last Modifed Date: Jan 4, 2006
Author Revision: 0.3

Number

OpenGL ES Extension #17 (formerly OpenGL Extension #295)

Dependencies

None
The extension is written against the OpenGL 1.3 Specification.

Overview

This extension provides the capability to query an OpenGL implementation for a preferred type and format combination for use with reading the color buffer with the ReadPixels command. The purpose is to enable embedded implementations to support a greatly reduced set of type/format combinations and provide a mechanism for applications to determine which implementation-specific combination is supported.

The preferred type and format combination returned may depend on the read surface bound to the current GL context.

IP Status

None

Issues

* Should this be generalized for other commands: DrawPixels, TexImage?

Resolved: No need to aggrandize.

New Procedures and Functions

None

New Tokens

IMPLEMENTATION_COLOR_READ_TYPE_OES	0x8B9A
IMPLEMENTATION_COLOR_READ_FORMAT_OES	0x8B9B

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

None

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

None

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

Section 4.3 Drawing, Reading, and Copying Pixels

Section 4.3.2 Reading Pixels

(add paragraph)
A single format and type combination, designated the preferred format, is associated with the state variables IMPLEMENTATION_COLOR_READ_FORMAT_OES and IMPLEMENTATION_COLOR_READ_TYPE_OES. The preferred format indicates a read format and type combination that provides optimal performance, for the read surface that is bound to the current GL context, for a particular implementation. The state values are chosen from the set of regularly accepted format and type parameters as shown in tables 3.6 and 3.5.

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

None

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

None

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance)

None

Additions to the AGL/GLX/WGL Specifications

None

Additions to the WGL Specification

None

Additions to the AGL Specification

None

Additions to Chapter 2 of the GLX 1.3 Specification (GLX Operation)

Additions to Chapter 3 of the GLX 1.3 Specification (Functions and Errors)

Additions to Chapter 4 of the GLX 1.3 Specification (Encoding on the X Byte Stream)

Additions to Chapter 5 of the GLX 1.3 Specification (Extending OpenGL)

Additions to Chapter 6 of the GLX 1.3 Specification (GLX Versions)

GLX Protocol

TBD

Errors

None

New State

None

New Implementation Dependent State

(table 6.28)

Get Value	Type	Get Command	Value	Description	Sec.	Attribute
x_FORMAT_OES	Z_11	GetIntegerv	-	read format	4.3.2	-
x_TYPE_OES	Z_20	GetIntegerv	-	read type	4.3.2	-

x_ = IMPLEMENTATION_COLOR_READ_

Revision History

```
    02/20/2003    0.1
        - Original draft.

    07/08/2003    0.2
        - Marked issue regarding extending to other commands to resolved.
        - Hackery to make state table fit in 80 columns
        - Removed Dependencies on section
        - Added extension number and enumerant values

    01/04/2006    0.3
        - Added clarification that format and type value returned
          depends on the current read surface attached to the current context
```

B.5 OES_query_matrix

Name

OES_query_matrix

Name Strings

GL_OES_query_matrix

Contact

Kari Pulli, Nokia (kari.pulli 'at' nokia.com)

Status

Ratified by the Khronos BOP, July 23, 2003.

Version

$Date: 2003/07/23 04:23:25 $ $Revision: 1.2 $

Number

OpenGL ES Extension #16 (formerly OpenGL Extension #296)

Dependencies

OpenGL 1.3 is required.
OES_fixed_point is required.

Overview

Many applications may need to query the contents and status of the current matrix at least for debugging purposes, especially as the implementations are allowed to implement matrix machinery either in any (possibly proprietary) floating point format, or in a fixed point format that has the range and accuracy of at least 16.16 (signed 16 bit integer part, unsigned 16 bit fractional part).

This extension is intended to allow application to query the components of the matrix and also their status, regardless whether the internal representation is in fixed point or floating point.

IP Status

There is no intellectual property associated with this extension.

Issues

None known.

New Procedures and Functions

```
GLbitfield glQueryMatrixxOES( GLfixed mantissa[16],
                              GLint   exponent[16] )
```

mantissa[16] contains the contents of the current matrix in GLfixed format. exponent[16] contains the unbiased exponents applied to the matrix components, so that the internal representation of component i is close to mantissa[i] * 2^exponent[i]. The function returns a status word which is zero if all the components are valid. If status & (1<<i) != 0, the component i is invalid (e.g., NaN, Inf). The implementations are not required to keep track of overflows. In that case, the invalid bits are never set.

New Tokens

None

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

None

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

None

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

None

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

None

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

Insert Overview and New Procedures and Functions to become Section 6.1.13.

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance)

None

Additions to the AGL/GLX/WGL Specifications

GLX Protocol

QueryMatrixxOES() is mapped to the equivalent protocol for floating-point state queries. Two queries are required; one to retrieve the current matrix mode and another to retrieve the matrix values.

Dependencies on OES_fixed_point

OES_fixed_point is required for the GLfixed definition.

Errors

None

New State

None

New Implementation Dependent State

None

Revision History

Apr 15, 2003	Kari Pulli	Created the document
Jul 08, 2003	David Blythe	Clarified the Dependencies section, Added extension number
Jul 12, 2003	David Blythe	Add GLX protocol note

B.6 OES_compressed_paletted_texture

Name

OES_compressed_paletted_texture

Name Strings

GL_OES_compressed_paletted_texture

Contact

Aaftab Munshi, ATI (amunshi@ati.com)

Notice

IP Status

No known IP issues

Status

Ratified by the Khronos BOP, July 23, 2003.

Version

Last Modifed Date: 12 November 2005
Author Revision: 0.6

Number

OpenGL ES Extension #6 (formerly OpenGL Extension #294)

Dependencies

Written based on the wording of the OpenGL ES 1.0 specification

Overview

The goal of this extension is to allow direct support of palettized textures in OpenGL ES.

Palettized textures are implemented in OpenGL ES using the CompressedTexImage2D call. The definition of the following parameters "level" and "internalformat" in the CompressedTexImage2D call have been extended to support paletted textures.

A paletted texture is described by the following data:

palette format
can be R5_G6_B5, RGBA4, RGB5_A1, RGB8, or RGBA8

number of bits to represent texture data

can be 4 bits or 8 bits per texel. The number of bits also detemine the size of the palette. For 4 bits/texel the palette size is 16 entries and for 8 bits/texel the palette size will be 256 entries.

The palette format and bits/texel are encoded in the "internalformat" parameter.

palette data and texture mip-levels

The palette data followed by all necessary mip levels are passed in "data" parameter of CompressedTexImage2D.

The size of palette is given by palette format and bits / texel. A palette format of RGB_565 with 4 bits/texel imply a palette size of 2 bytes/palette entry * 16 entries = 32 bytes.

The level value is used to indicate how many mip levels are described. Negative level values are used to define the number of miplevels described in the "data" component. A level of zero indicates a single mip-level.

Issues

* Should glCompressedTexSubImage2D be allowed for modifying paletted texture data.

  RESOLVED: No, this would then require implementations that do not support paletted formats internally to also store the palette per texture. This can be a memory overhead on platforms that are memory constrained.

* Should palette format and number of bits used to represent each texel be part of data or internal format.

  RESOLVED: Should be part of the internal format since this makes the palette format and texture data size very explicit for the application programmer.

* Should the size of palette be fixed i.e 16 entries for 4-bit texels and 256 entries for 8-bit texels or be programmable.

  RESOLVED: Should be fixed. The application can expand the palette to 16 or 256 if internally it is using a smaller palette.

New Procedures and Functions

None

New Tokens

Accepted by the <level> parameter of CompressedTexImage2D

Zero and negative values. |level| + 1 determines the number of mip levels defined for the paletted texture.

Accepted by the <internalformat> paramter of CompressedTexImage2D

PALETTE4_RGB8_OES	0x8B90
PALETTE4_RGBA8_OES	0x8B91
PALETTE4_R5_G6_B5_OES	0x8B92
PALETTE4_RGBA4_OES	0x8B93
PALETTE4_RGB5_A1_OES	0x8B94
PALETTE8_RGB8_OES	0x8B95
PALETTE8_RGBA8_OES	0x8B96
PALETTE8_R5_G6_B5_OES	0x8B97
PALETTE8_RGBA4_OES	0x8B98
PALETTE8_RGB5_A1_OES	0x8B99

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

None

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

Add to Table 3.17: Specific Compressed Internal Formats

Compressed Internal Format	Base Internal Format
PALETTE4_RGB8_OES	RGB
PALETTE4_RGBA8_OES	RGBA
PALETTE4_R5_G6_B5_OES	RGB
PALETTE4_RGBA4_OES	RGBA
PALETTE4_RGB5_A1_OES	RGBA
PALETTE8_RGB8_OES	RGB
PALETTE8_RGBA8_OES	RGBA
PALETTE8_R5_G6_B5_OES	RGB
PALETTE8_RGBA4_OES	RGBA
PALETTE8_RGB5_A1_OES	RGBA

Add to Section 3.8.3, Alternate Image Specification

If <internalformat> is PALETTE4_RGB8, PALETTE4_RGBA8, PALETTE4_R5_G6_B5, PALETTE4_RGBA4, PALETTE4_RGB5_A1, PALETTE8_RGB8, PALETTE8_RGBA8, PALETTE8_R5_G6_B5, PALETTE8_RGBA4 or PALETTE8_RGB5_A1, the compressed texture is a compressed paletted texture. The texture data contains the palette data following by the mip-levels where the number of mip-levels stored is given by |level| + 1. The number of bits that represent a texel is 4 bits if <interalformat> is given by PALETTE4_xxx and is 8 bits if <internalformat> is given by PALETTE8_xxx.

The number of bits that represent each palette entry is:

Compressed Internal Format	# of bits / palette entry
PALETTE4_RGB8_OES	24
PALETTE4_RGBA8_OES	32
PALETTE4_R5_G6_B5_OES	16
PALETTE4_RGBA4_OES	16
PALETTE4_RGB5_A1_OES	16
PALETTE8_RGB8_OES	24
PALETTE8_RGBA8_OES	32
PALETTE8_R5_G6_B5_OES	16
PALETTE8_RGBA4_OES	16
PALETTE8_RGB5_A1_OES	16

Compressed paletted textures support only 2D images without borders. CompressedTexImage2D will produce an INVALID_OPERATION error if <border> is non-zero.

To determine palette format refer to tables 3.10 and 3.11 of Chapter 3 where the data ordering for different <type> formats are described.

Add table 3.17.1: Texel Data Formats for compressed paletted textures

PALETTE4_xxx:

```
 7 6 5 4 3 2 1 0
 ---------------
|  1st  |  2nd  |
| texel | texel |
 ---------------
```

PALETTE8_xxx

```
 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 ---------------------------------------------------------------------------------------
|         4th             |          3nd            |          2rd         |     1st
|        texel            |         texel           |         texel        |    texel
 ---------------------------------------------------------------------------------------
```

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

None

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

None

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

None

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance)

Additions to the AGL/GLX/WGL Specification

None

GLX Protocol

None

Errors

INVALID_OPERATION is generated by TexImage2D, CompressedTexSubImage2D, CopyTexSubImage2D if <internalformat> is PALETTE4_RGB8_OES, PALETTE4_RGBA8_OES, PALETTE4_R5_G6_B5_OES, PALETTE4_RGBA4_OES, PALETTE4_RGB5_A1_OES, PALETTE8_RG8_OES, PALETTE8_RGBA8_OES, PALETTE8_R5_G6_B5_OES, PALETTE8_RGBA4_OES, or PALETTE8_RGB5_A1_OES.

INVALID_VALUE is generated by CompressedTexImage2D if if <internalformat> is 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, or PALETTE8_RGB5_A1_OES and <level> value is neither zero or a negative value.

New State

The queries for NUM_COMPRESSED_TEXTURE_FORMATS and COMPRESSED_TEXTURE_FORMATS include these ten new formats.

Revision History

04/28/2003 0.1 (Aaftab Munshi)
- Original draft.

05/29/2003 0.2 (David Blythe)
- Use paletted rather than palettized. Change naming of internal

```
           format tokens to match scheme used for other internal formats.

    07/08/2003    0.3    (David Blythe)
         - Add official enumerant values and extension number.

    07/09/2003    0.4    (David Blythe)
         - Note that [NUM_]COMPRESSED_TEXTURE_FORMAT queries include the
           new formats.

    07/21/2004    0.5    (Aaftab Munshi)
   - Fixed PALETTE_8xxx drawing

    11/12/2005    0.6    (Aaftab Munshi)
       - Corrections
```

B.7 OES_matrix_palette

Name

OES_matrix_palette

Name Strings

GL_OES_matrix_palette

Contact

Aaftab Munshi (amunshi@ati.com)

Status

Ratified by the Khronos BOP, Aug 5, 2004.

Version

Number

OpenGL ES Extension #12

Dependencies

OpenGL ES 1.0 is required.

Overview

This extension adds the ability to support vertex skinning in OpenGL ES. A simplified version of the ARB_matrix_palette extension is used to define OES_matrix_palette extension.

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.

A similar procedure is followed for normals. Normals, however, are transformed by the inverse transpose of the modelview matrix.

IP Status

Unknown, but believed to be none.

Issues

Should this extension be an optional or mandatory extension

Will be an optional extension since ARB_matrix_palette didn’t see much usage in OpenGL.

Should we allow the ability to load the current model view matrix into the matrix palette

Yes. This will be very helpful since it makes it very easy to load an object heirarchy. This will also be helpful for JSR184

Should the Matrix palette be loaded with a new LoadMatrixPalette command?

No, although this provides an easy way to support arbitrary palette sizes, the method loses the current (MultMatrix, Rotate, Translate, Scale..) matrix functionality.

Matrices will be Loaded into the palette with current functions when MATRIX_MODE is MATRIX_PALETTE_OES. The current palette index is set by an explicit command: CurrentPaletteMatrixARB().

Should the Matrix Palette have a stack?

Not required, this wastes a lot of space.

Should the matrix palette be gettable?

No.

Should MatrixIndexARB be changed to imply LoadMatrix calls to the applicable MODELVIEW_MATRIXn stacks?

No, the MODELVIEW_MATRIXn matrices are unused when MATRIX_PALETTE is enabled.

Should there be a way to specify that the modelview matrices for two different vertex units are identical?

Not explicitly, but indexing the matrix palette provides this functionality. (Both units will have the same matrix index.)

New Procedures and Functions

```
void CurrentPaletteMatrixOES(uint index)

void LoadPaletteFromModelViewMatrixOES()

void MatrixIndexPointerOES(int size, enum type, sizei stride, void *pointer)

void WeightPointerOES(int size, enum type, sizei stride, void *pointer);
```

New Tokens

Accepted by the <mode> parameter of MatrixMode, and by the <cap> parameters of Enable and Disable:

MATRIX_PALETTE_OES	0x8840

Accepted by the <pname> parameters of GetIntegerv:

MAX_PALETTE_MATRICES_OES	0x8842
MAX_VERTEX_UNITS_OES	0x86A4
CURRENT_PALETTE_MATRIX_OES	0x8843

The default values for MAX_PALETTE_MATRICES_OES and MAX_VERTEX_UNITS_OES are 9 and 3 resp.

Accepted by the <cap> parameters of EnableClientState and DisableClientState and by the <pname> parameter of IsEnabled:

MATRIX_INDEX_ARRAY_OES	0x8844
WEIGHT_ARRAY_OES	0x86AD

Accepted by the <pname> parameter of GetIntegerv:

MATRIX_INDEX_ARRAY_SIZE_OES	0x8846
MATRIX_INDEX_ARRAY_TYPE_OES	0x8847
MATRIX_INDEX_ARRAY_STRIDE_OES	0x8848
MATRIX_INDEX_ARRAY_BUFFER_BINDING_OES	0x8B9E
WEIGHT_ARRAY_SIZE_OES	0x86AB
WEIGHT_ARRAY_TYPE_OES	0x86A9
WEIGHT_ARRAY_STRIDE_OES	0x86AA
WEIGHT_ARRAY_BUFFER_BINDING_OES	0x889E

Accepted by the <pname> parameter of GetPointerv:

MATRIX_INDEX_ARRAY_POINTER_OES	0x8849
WEIGHT_ARRAY_POINTER_OES	0x86AC

Additions to Chapter 2 of the OpenGL ES 1.0 Specification

- Added to section 2.8

```
void WeightPointerOES(int size, enum type, sizei stride, void *pointer);

void MatrixIndexPointerOES(int size, enum type, sizei stride, void *pointer);
```

WeightPointerOES & MatrixIndexPointerOES are used to describe the weights and matrix indices used to blend corresponding matrices for a given vertex.

For implementations supporting matrix palette, note that <size> values for WeightPointerOES & MatrixIndexPointerOES must be less than or equal to the implementation defined value MAX_VERTEX_UNITS_OES.

- Added to table in section 2.8

Command	Sizes	Types
WeightPointerOES	1..MAX_VERTEX_UNITS_OES	fixed, float
MatrixIndexPointerOES	1..MAX_VERTEX_UNITS_OES	ubyte

- (section 2.8) Extend the cap flags passed to EnableClientState/DisableClientState to include

  MATRIX_INDEX_ARRAY_OES, or WEIGHT_ARRAY_OES

- (section 2.10) Add the following:

"The vertex coordinates that are presented to the GL are termed object coordinates. The model-view matrix is applied to these coordinates to yield eye coordinates. In implementations with matrix palette, the matrices specified by the indices per vertex are applied to these coordinates and the weighted sum of the results are the eye coordinates. Then another matrix, called the projection matrix, is applied to eye coordinates to yield clip coordinates. A perspective division is carried out on clip coordinates to yield normalized device coordinates.

A final viewport transformation is applied to convert these coordinates into window coordinates."

"... the vertex's eye coordinates are found as:

```
(xe)    n-1                (xo)
(ye)  =  SUM  w_i * M_i * (yo)
(ze)    i=0                (zo)
(we)                       (wo)
```

where M_i is the palette matrix associated with the i'th Vertex unit:

```
  M_i = MatrixPalette[MatrixIndex[i]],
           if MATRIX_PALETTE_OES is enabled, and
  M_i = MODELVIEW_MATRIX, otherwise.
```

w_i is the Vertex’s associated weight for vertex unit i:

```
  w_i = weight_i, if MATRIX_PALETTE_OES is enabled,
               1, if MATRIX_PALETTE_OES is disabled,
```

and,

```
  n = <size> value passed into glMatrixIndexPointerOES."
```

"The projection matrix and model-view matrices are set with a variety of commands. The affected matrix is determined by the current matrix mode. The current matrix mode is set with

```
  void MatrixMode( enum mode );
```

which takes one of the pre-defined constants TEXTURE, MODELVIEW, PROJECTION, MATRIX_PALETTE_OES.

In implementations supporting OES_matrix_palette,

```
   void CurrentPaletteMatrixOES(uint index);
```

defines which of the palette’s matrices is affected by subsequent matrix operations when the current matrix mode is MATRIX_PALETTE_OES. CurrentPaletteMatrixOES generates the error INVALID_VALUE if the <index> parameter is not between 0 and MAX_PALETTE_MATRICES_OES - 1.

In implementations supporting OES_matrix_palette,

```
   void LoadPaletteFromModelViewMatrixOES();
```

copies the current model view matrix to a matrix in the matrix palette, specified by CurrentPaletteMatrixOES.

DrawArrays and DrawElements will not render the primitive if the matrix palette was enabled and the weights and/or matrix index vertex pointers are disabled or are not valid.

"The state required to implement transformations consists of a four-valued integer indicating the current matrix mode, a stack of at least two 4 x 4 matrices for each of PROJECTION, and TEXTURE with associated stack pointers, a stack of at least 32 4 x 4 matrices with an associated stack pointer for MODELVIEW, and a set of MAX_PALETTE_MATRICES_OES matrices of at least 9 4 x 4 matrices each for the matrix palette.

Initially, there is only one matrix on each stack, and all matrices are set to the identity. The initial matrix mode is MODELVIEW.

"When matrix palette is enabled, the normal is transformed to eye space by:

```
                                    n-1
(nx' ny' nz') = (nx ny nz) Inv ( SUM w_i * Mu_i)
                                    i=0
```

Alternatively implementations may choose to transform the normal to eye-space by:

```
                    n-1
(nx' ny' nz') =  SUM w_i * (nx ny nz) Inv(Mu_i)
                    i=0
```

where Mu_i is the upper leftmost 3x3 matrix taken from the modelview for vertex unit i (M_i),

```
M_i = MatrixPalette[MatrixIndex[i]],
          if MATRIX_PALETTE_OES is enabled, and
M_i = MODELVIEW_MATRIX, otherwise
```

otherwise.

weight_i is the vertex's associated weight for vertex unit i,

```
w_i = weight_i
```

and

```
n = <size> value passed into glMatrixIndexPointerOES."
```

Errors

INVALID_VALUE is generated if the <size> parameter for MatrixIndexPointerOES or WeightPointerOES is greater than MAX_VERTEX_UNITS_OES.

INVALID_VALUE is generated if the <count> parameter to CurrentPaletteMatrixOES is greater than MAX_PALETTE_MATRICES_OES - 1

New State

(table 6.6, p. 232)

Get Value	Type	Get Command	Initial Value	Description
MATRIX_INDEX_ARRAY_OES	B	IsEnabled	False	matrix index array enable
MATRIX_INDEX_ARRAY_SIZE_OES	Z+	GetIntegerv	0	matrix indices per vertex
MATRIX_INDEX_ARRAY_TYPE_OES	Z+	GetIntegerv	UBYTE	type of matrix index data
MATRIX_INDEX_ARRAY_STRIDE_OES	Z+	GetIntegerv	0	stride between matrix indices
MATRIX_INDEX_ARRAY_POINTER_OES	Y	GetPointerv	0	pointer to matrix index array
WEIGHT_ARRAY_OES	B	IsEnabled	False	weight array enable
WEIGHT_ARRAY_SIZE_OES	Z+	GetIntegerv	0	weights per vertex
WEIGHT_ARRAY_TYPE_OES	Z2	GetIntegerv	FLOAT	type of weight data
WEIGHT_ARRAY_STRIDE_OES	Z+	GetIntegerv	0	stride between weights per vertex
WEIGHT_ARRAY_POINTER_OES	Y	GetPointerv	0	pointer to weight array

(table 6.7, p. 233)

Get Value	Type	Get Command	Initial Value	Description
MATRIX_INDEX_ARRAY_BUFFER_BINDING_OES	Z+	GetIntegerv	0	matrix index array buffer binding
WEIGHT_ARRAY_BUFFER_BINDING_OES	Z+	GetIntegerv	0	weight array buffer binding

(table 6.9, p. 235)

Get Value	Type	Get Command	Initial Value	Description
MATRIX_PALETTE_OES	B	IsEnabled	False	matrix palette enable
MAX_PALETTE_MATRICES_OES	Z+	GetIntegerv	9	size of matrix palette
MAX_VERTEX_UNITS_OES	Z+	GetIntegerv	3	number of matrices per vertex
CURRENT_PALETTE_MATRIX_OES	Z+	GetIntegerv	0	transform index of current modelview matrix in the palette, as set by CurrentPaletteMatrixOES()

Revision History

Addendum: Using this extension.

```
/* position viewer */
glMatrixMode(GL_MATRIX_PALETTE_OES);
glCurrentPaletteMatrixOES(0);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -7.0f);
glRotatef(yrot, 0.0f, 1.0f, 0.0f);

glCurrentPaletteMatrixOES(1);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -7.0f);

glRotatef(yrot, 0.0f, 1.0f, 0.0f);
glRotatef(zrot, 0.0f, 0.0f, 1.0f);

glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glEnableClientState(GL_MATRIX_INDEX_ARRAY_OES);
glEnableClientState(GL_WEIGHT_ARRAY_OES);

glVertexPointer(3, GL_FLOAT, 7 * sizeof(GLfloat), vertexdata);
glTexCoordPointer(2, GL_FLOAT, 7 * sizeof(GLfloat), vertexdata + 3);
glWeightPointerOES(2, GL_FLOAT, 7 * sizeof(GLfloat),vertexdata + 5);
glMatrixIndexPointerOES(2, GL_UNSIGNED_BYTE, 0, matrixindexdata);

for(int i = 0; i < (numSegments << 2) + 2; i ++)
    glDrawArrays(GL_TRIANGLE_FAN, i << 2, 4);
```

B.8 OES_point_sprite

Name

OES_point_sprite

Name Strings

GL_OES_point_sprite

Contact

Aaftab Munshi (amunshi@ati.com)

Status

Ratified by the Khronos BOP, Aug 5, 2004.

IP Status

No known IP issues.

Version

Last Modified Date: August 5, 2004

Number

OpenGL ES Extension #15

Dependencies

OpenGL ES 1.0 is required

Overview

Applications such as particle systems have tended to use OpenGL quads rather than points to render their geometry, since they would like to use a custom-drawn texture for each particle, rather than the traditional OpenGL round antialiased points, and each fragment in a point has the same texture coordinates as every other fragment.

Unfortunately, specifying the geometry for these quads can be expensive, since it quadruples the amount of geometry required, and may also require the application to do extra processing to compute the location of each vertex.

The purpose of this extension is to allow such applications to use points rather than quads. When GL_POINT_SPRITE_OES is enabled, the state of point antialiasing is ignored. For each texture unit, the app can then specify whether to replace the existing texture coordinates with point sprite texture coordinates, which are interpolated across the point.

Issues

The following are the list of issues as discussed in the ARB_point_sprite extension. I've deleted issues that are not related to OpenGL ES such as vertex shader programs etc.

Tokens that use _ARB names are modified to use _OES.

* Should this spec say that point sprites get converted into quads?

RESOLVED: No, this would make the spec much uglier, because then we'd have to say that polygon smooth and stipple get turned off, etc. Better to provide a formula for computing the texture coordinates and leave them as points.

* How are point sprite texture coordinates computed?

RESOLVED: They move smoothly as the point moves around on the screen, even though the pixels touched by the point do not. The exact formula is given in the spec below.

A point sprite can be thought of as a quad whose upper-left corner has (s,t) texture coordinates of (0,0) and whose lower-right corner has texture coordinates of (1,1), as illustrated in the following figure. In the figure "P" is the center of the point sprite, and "O" is the origin (0,0) of the window coordinate system. Note that the y window coordinate increases from bottom-to-top but the t texture coordinate of point sprites increases from top-to-bottom.

```
     ^
   +y| (0,0)
     |   +-----+
     |   |     |
     |   |  P  |
     |   |     |
     |   +-----+
     |       (1,1)
     |              +x
     O--------------->
```

Applications using a single texture for both point sprites and other geometry need to account for the fixed coordinate mapping of point sprites.

* How do point sizes for point sprites work?

RESOLVED: This specification treats point sprite sizes like antialiased point sizes, but with more leniency. Implementations may choose to not clamp the point size to the antialiased point size range. The set of point sprite sizes available must be a superset of the antialiased point sizes. However, whereas antialiased point sizes are all evenly spaced by the point size granularity, point sprites can have an arbitrary set of sizes. This lets implementations use, e.g., floating-point sizes.

- Should there be a way to query the list of supported point sprite sizes?

  RESOLVED: No. If an implementation were to use, say, a single-precision IEEE float to represent point sizes, the list would be rather long.

- Do mipmaps apply to point sprites?

  RESOLVED: Yes. They are similar to quads in this respect.

- What of this extension's state is per-texture unit and what of this extension's state is state is global?

  RESOLVED: The GL_POINT_SPRITE_OES enable is global. The COORD_REPLACE_OES state is per-texture unit (state set by TexEnv is per-texture unit).

- Should there be a global on/off switch for point sprites, or should the per-unit enable imply that switch?

  RESOLVED: There is a global switch to turn it on and off. This is probably more convenient for both driver and app, and it simplifies the spec.

- What should the TexEnv mode for point sprites be called?

  RESOLVED: COORD_REPLACE_OES.

- What is the interaction with multisample points, which are round?

  RESOLVED: Point sprites are rasterized as squares, even in multisample mode. Leaving them as round points would make the feature useless.

- How does this extension interact with the point size attenuation functionality in OES_point_parameters and OpenGL 1.4?

  RESOLVED: Point sprites sizes are attenuated just like the sizes of non-sprite points.

- How are point sprites clipped?

  RESOLVED: Point sprites are transformed as points, and standard point clipping operations are performed. This can cause point sprites that move off the edge of the screen to disappear abruptly, in the same way that regular points do. As with any other primitive, standard per-fragment clipping operations (scissoring, window ownership test) still apply.

New Procedures and Functions

None

New Tokens

Accepted by the <cap> parameter of Enable, Disable, and by the <target> parameter of TexEnvf, TexEnvfv, TexEnvx, TexEnvxv:

```
    POINT_SPRITE_OES                              0x8861
```

When the <target> parameter of TexEnvf, TexEnvfv, TexEnvx, TexEnvxv, is POINT_SPRITE_OES, then the value of <pname> may be:

```
    COORD_REPLACE_OES                             0x8862
```

When the <target> and <pname> parameters of TexEnvf, TexEnvfv, TexEnvx, TexEnvxv, are POINT_SPRITE_OES and COORD_REPLACE_OES respectively, then the value of <param> or the value pointed to by <params> may be:

```
    FALSE
    TRUE
```

Additions to Chapter 2 of the OpenGL 1.4 Specification (OpenGL Operation)

None.

Additions to Chapter 3 of the OpenGL 1.4 Specification (Rasterization)

Insert the following paragraphs after the second paragraph of section 3.3 (page 66):

"Point sprites are enabled or disabled by calling Enable or Disable with the symbolic constant POINT_SPRITE_OES. The default state is for point sprites to be disabled. When point sprites are enabled, the state of the point antialiasing enable is ignored.

The point sprite texture coordinate replacement mode is set with one of the commands

```
  void TexEnv{ixf}(enum target, enum pname, T param)
  void TexEnv{ixf}v(enum target, enum pname, const T *params)
```

where target is POINT_SPRITE_OES and pname is COORD_REPLACE_OES. The possible values for param are FALSE and TRUE. The default value for each texture unit is for point sprite texture coordinate replacement to be disabled."

Replace the first two sentences of the second paragraph of section 3.3.1 (page 67) with the following:

"The effect of a point width other than 1.0 depends on the state of point antialiasing and point sprites. If antialiasing and point sprites are disabled, ..."

Replace the first sentences of the fourth paragraph of section 3.3.1 (page 68) with the following:

"If antialiasing is enabled and point sprites are disabled, ..."

Insert the following paragraphs at the end of section 3.3.1 (page 70):

"When point sprites are enabled, then point rasterization produces a fragment for each framebuffer pixel whose center lies inside a square centered at the point's (x_w, y_w), with side length equal to the current point size.

All fragments produced in rasterizing a point sprite are assigned the same associated data, which are those of the vertex corresponding to the point, with texture coordinates s, t, and r replaced with s/q, t/q, and r/q, respectively. If q is less than or equal to zero, the results are undefined. However, for each texture unit where COORD_REPLACE_OES is TRUE, these texture coordinates are replaced with point sprite texture coordinates. The s coordinate varies from 0 to 1 across the point horizontally left-to-right, while the t coordinate varies from 0 to 1 vertically top-to-bottom. The r and q coordinates are replaced with the constants 0 and 1, respectively.

The following formula is used to evaluate the s and t coordinates:

```
  s = 1/2 + (x_f + 1/2 - x_w) / size
  t = 1/2 - (y_f + 1/2 - y_w) / size
```

where size is the point's size, x_f and y_f are the (integral) window coordinates of the fragment, and x_w and y_w are the exact, unrounded window coordinates of the vertex for the point.

The widths supported for point sprites must be a superset of those supported for antialiased points. There is no requirement that these widths must be equally spaced. If an unsupported width is requested, the nearest supported width is used instead."

Replace the text of section 3.3.2 (page 70) with the following:

"The state required to control point rasterization consists of the floating-point point width, three floating-point values specifying the minimum and maximum point size and the point fade threshold size, three floating-point values specifying the distance attenuation coefficients, a bit indicating whether or not antialiasing is enabled, a bit indicating whether or not point sprites are enabled, and a bit for the point sprite texture coordinate replacement mode for each texture unit."

Replace the text of section 3.3.3 (page 70) with the following:

"If MULTISAMPLE is enabled, and the value of SAMPLE_BUFFERS is one, then points are rasterized using the following algorithm, regardless of whether point antialiasing (POINT_SMOOTH) is enabled or disabled. Point rasterization produces a fragment for each framebuffer pixel with one or more sample points that intersect a region centered at the point's (x_w, y_w). This region is a circle having diameter equal to the current point width if POINT_SPRITE_OES is disabled, or a square with side equal to the current point width if POINT_SPRITE_OES is enabled. Coverage bits that correspond to sample points that intersect the region are 1, other coverage bits are 0. All data associated with each sample for the fragment are the data associated with the point being rasterized, with the exception of texture coordinates when POINT_SPRITE_OES is enabled; these texture coordinates are computed as described in section 3.3.

Point size range and number of gradations are equivalent to those supported for antialiased points when POINT_SPRITE_OES is disabled. The set of point sizes supported is equivalent to those for point sprites without multisample when POINT_SPRITE_OES is enabled."

Additions to Chapter 4 of the OpenGL 1.4 Specification (Per-Fragment Operations and the Frame Buffer)

None.

Additions to Chapter 5 of the OpenGL 1.4 Specification (Special Functions)

None.

Additions to Chapter 6 of the OpenGL 1.4 Specification (State and State Requests)

None.

Errors

None.

New State

(table 6.12, p. 220)

Get Value	Type	Get Command	Initial Value	Description

POINT_SPRITE_OES	B	IsEnabled	False	point sprite enable

(table 6.17, p. 225)

Get Value	Type	Get Command	Initial Value	Description
COORD_REPLACE_OES	2* x B	GetTexEnviv	False	coordinate replacement enable

Revision History

B.9 OES_point_size_array

Name

OES_point_size_array

Name Strings

GL_OES_point_size_array

Contact

Aaftab Munshi (amunshi@ati.com)

Status

Ratified by the Khronos BOP, Aug 5, 2004.

Version

Last Modifed Date: 17 Dec 2004

Number

OpenGL ES Extension #14

Dependencies

OpenGL ES 1.0 is required.
OES_point_sprite is required

The extension is written against the OpenGL 1.5 Specification.

Overview

This 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.

IP Status

None.

Issues

New Procedures and Functions

```
void PointSizePointerOES(enum type, sizei stride, const void *ptr )
```

valid values of type are GL_FIXED and GL_FLOAT
the <size> parameter is removed since <size> is always 1

New Tokens

Accepted by the <cap> parameters of EnableClientState/DisableClientState and by the <pname> parameter of IsEnabled:

POINT_SIZE_ARRAY_OES	0x8B9C

Accepted by the <pname> parameter of GetIntegerv:

POINT_SIZE_ARRAY_TYPE_OES	0x898A
POINT_SIZE_ARRAY_STRIDE_OES	0x898B
POINT_SIZE_ARRAY_BUFFER_BINDING_OES	0x8B9F

Accepted by the <pname> parameter of GetPointerv:

POINT_SIZE_ARRAY_POINTER_OES	0x898C

Additions to Chapter 2 of the OpenGL 1.5 specification

- section 2.8, added the following

```
void PointSizePointerOES(enum type, sizei stride, const void *ptr);
```

PointSizePointerOES is used to describe the point size for a given vertex

- Added to table 2.4

Command	Sizes	Types
PointSizePointerOES	1	float, fixed

- (section 2.8), added the following
  Extend the cap flags passed to EnableClientState/DisableClientState to include POINT_SIZE_ARRAY_OES

If point size array is enabled but the point size vertex pointers are invalid, then DrawArrays and DrawElements will not render the point primitive.

Errors

None.

New State

(table 6.6, p. 232)

Get Value	Type	Get Command	Initial Value	Description
POINT_SIZE_ARRAY_OES	B	IsEnabled	False	point sprite array enable
POINT_SIZE_ARRAY_TYPE_OES	Z2	GetIntegerv	Float	type of point size
POINT_SIZE_ARRAY_STRIDE_OES	Z+	GetIntegerv	0	stride between point sizes
POINT_SIZE_ARRAY_POINTER_OES	Y	GetPointerv	0	pointer to point sprite array

(table 6.7, p. 233)

Get Value	Type	Get Command	Initial Value	Description
POINT_SIZE_ARRAY_BUFFER_BINDING_OES	Z+	GetIntegerv	0	point size array buffer binding

Revision History

B.10 OES_matrix_get

Name

OES_matrix_get

Name Strings

GL_OES_matrix_get

Contact

Aaftab Munshi (amunshi@ati.com)

Status

Ratified by the Khronos BOP, Aug 5, 2004.

Version

Last Modified Date: July 16, 2004

Number

OpenGL ES Extension #11

Dependencies

OpenGL 1.5 is required

Overview

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 pt (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.

Bit 31 represents the sign of the floating pt number.

Bits 30 - 23 represent the exponent of the floating pt number.
Bits 22 - 0 represent the mantissa of the floating pt number.

IP Status

There is no intellectual property associated with this extension.

Issues

None known.

New Procedures and Functions

New Tokens

Accepted by the <pname> parameter of GetIntegerv:

MODELVIEW_MATRIX_FLOAT_AS_INT_BITS_OES 0x898d
PROJECTION_MATRIX_FLOAT_AS_INT_BITS_OES 0x898e
TEXTURE_MATRIX_FLOAT_AS_INT_BITS_OES 0x898f

Additions to Chapter 2 of the OpenGL 1.4 Specification (OpenGL Operation)

None.

Additions to Chapter 3 of the OpenGL 1.4 Specification (Rasterization)

None.

Additions to Chapter 4 of the OpenGL 1.4 Specification (Per-Fragment Operations and the Frame Buffer)

None.

Additions to Chapter 5 of the OpenGL 1.4 Specification (Special Functions)

None.

Additions to Chapter 6 of the OpenGL 1.4 Specification (State and State Requests)

The new matrix tokens return the matrix elements as exponent and mantissa terms. 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.

Errors

None.

New State

Get Value	Type	Command	Value
MODELVIEW_MATRIX_FLOAT_AS_INT_BITS_OES	4* x 4* x Z	GetIntegerv	0
PROJECTION_MATRIX_FLOAT_AS_INT_BITS_OES	4* x 4* x Z	GetIntegerv	0
TEXTURE_MATRIX_FLOAT_AS_INT_BITS_OES	4* x 4* x Z	GetIntegerv	0

Revision History

June 30, 2004	Aaftab Munshi	Initial version of document
July 16, 2004	Aaftab Munshi	Removed the description of NaN & denorms

B.11 OES_draw_texture

Name

OES_draw_texture

Name Strings

GL_OES_draw_texture

Contact

Tom Olson (t-olson ’at’ ti.com)

Status

Ratified by the Khronos BOP, Aug 5, 2004.

Version

Last Modified Date: 21 July 2004
Author Revision 0.96

Number

OpenGL ES Extension #7

Dependencies

OES_fixed_point is required.
EXT_fog_coord affects the definition of this extension.
This extension is written against the OpenGL 1.3 and OpenGL ES 1.0 Specifications.

Overview

This 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. This extension is primarily intended for use with OpenGL ES.

The extension relies on a new piece of texture state called the texture crop rectangle, which defines a rectangular subregion of a texture object. These subregions are used as sources of pixels for the texture drawing function.

Applications use this extension by configuring the texture crop rectangle for one or more textures via ActiveTexture() and TexParameteriv() with pname equal to

TEXTURE_CROP_RECT_OES. They then request a drawing operation using DrawTex{sifx}[v]OES(). The effect of the latter function is to generate a screen-aligned target rectangle, with texture coordinates chosen to map the texture crop rectangle(s) linearly to fragments in the target rectangle. The fragments are then processed in accordance with the fragment pipeline state.

IP Status

No known IP issues.

Issues

(1) Should we pass a texture name to the draw function, or use the currently bound texture?

RESOLVED. Use the textures bound to currently enabled texture units. This makes it easy for drivers to implement DrawTex*() using existing texture hardware. If we didn’t do this, they would have to save and restore the state of the texture unit(s).

(2) Doesn’t DrawPixels make this extension unnecessary?

RESOLVED. No. DrawPixels is hard to support efficiently in hardware because the source pixels are in application memory. Also, the pixel setup pipeline (PixelTransfer, PixelMap etc.) is redundant for the intended applications. Also, PixelZoom looks ugly when the zoom factors are large, and there is no way to control filtering. Using textures and texture units solves all of these problems.

(3) Doesn’t ARB_point_sprite make this extension unnecessary?

RESOLVED. No. Key differences include:

* ARB_point_sprite uses the entire source texture to paint a point, i.e. its texture coordinates range from 0.0 to 1.0. This extension allows a subregion of a texture to be used as the source.
* ARB_point_sprite sprites are limited by the maximum point size, which may be small. This extension is limited only by the maximum supported texture size and the screen size.
* ARB_point_sprite sprites are square. This extension supports general rectangles as sprite shapes.
* ARB_point_sprite sprites are clipped as points, so if the center of a sprite falls outside the frustrum, nothing is drawn. This extension draws

any portion of a sprite that lies within the viewing frustrum. (There is a well-known work-around for this, but it's ugly.)

(4) How is the texture sampled?

RESOLVED. It is sampled like a normal texture, and not like an image sent to DrawPixels. This facilitates implementing with texture hardware.

(5) How does this work when multisampling is enabled?

RESOLVED. Implementations should generate multisample texture coordinates using the same method they use in normal texture mapping. Approximations are acceptable, e.g. they may use the same texture value for all samples associated with a fragment generated by DrawTex*(), even if they use another policy for multisampled triangle rendering.

(6) Do we really want the full fragment pipeline to be active?

RESOLVED. Yes, on grounds of orthogonality and simplicity. Again, this makes it easy for existing hardware to implement the extension.

(7) How does this interact with user clip planes?

RESOLVED. User clip planes are ignored. This is a screen-level operation, so geometric entities like clip planes are irrelevant.

(8) How does this interact with mip-mapping?

RESOLVED. It behaves exactly as in texturing. This is really easy to do as LOD is a constant across the target rectangle.

(9) What happens when multiple texture units are enabled?

RESOLVED. All enabled texture units participate in generating the final fragment color. Each unit generates its own s,t based on its texture's crop rectangle.

(10) Should the target location be specified by the current raster position (RasterPos or WindowPos), or by arguments to DrawTex*OES()?

RESOLVED. Use arguments passed to DrawTex*. In the intended uses, the target will be set once per call, so using arguments saves one inner loop function call.

(11) Do we want stretch-blt capability?

RESOLVED. Yes. Supply a window size as well as window position to DrawTex*()

(12) OpenGL ES issue: WindowPos (if we use it) adds 16 entry points ({23}{sifd}[v]), which seems like a lot even if they are trivial. Can we live with a subset? (Note that the ’d’ versions go away, but they are replaced by ’x’ versions.)

RESOLVED. Moot, as we do not use WindowPos. But the intent was to add only 3{si}[v] versions (four entry points). This is not orthogonal and may be surprising. But there is no intent to support sub-pixel placement of rectangles, so the {fx} versions are superfluous. {2} versions are easy to express using {3} versions. Vector and individual argument versions are kept to reduce the surprise factor, and because constructing calls to a v-type function is a huge pain if you don’t already have the data in vector format.

(13) TexCropRect*OES adds eight entry points. Can we live with a subset? For the intended use, integer values suffice, so the {fx} versions are superfluous. But orthogonality and ’least-astonishment’ are virtues too.

RESOLVED. Moot. Replace with TexParameteriv().

(14) Would it be better to remove the texture crop rectangle from the state, and instead pass parameters to DrawTextureOES()?

RESOLVED. No. Drawing the same pixel pattern multiple times is a plausible usage scenario.

(15) Should texture crop rect parameters be stored internally as integers, or as float/fixed? I.e. should we allow the crop rect to include fractional texels? This is more flexible, but is not the intended use. Software implementations would have to add a test for the (normal) special case of integer bounds.

RESOLVED. Integer only. Texture crop rect is conceptually a subregion of an integer grid, so its natural coordinates are integers.

(16) Should we have a single global crop rect, or one per texture unit?

RESOLVED. Neither. We should have one per texture, with TexParameter setting the rect for the currently active texture. It isn't a lot of state, it attaches the rect to a specific texture (which makes sense) rather than a texture unit (which doesn't), it is more orthogonal, and it allows tex coords to be meaningful (if not actually useful) when multiple texture units are enabled.

(17) Should the destination rectangle specified by DrawTex*() be defined as integer only like the crop rectangle, or should its parameters be real-valued?

RESOLVED. Real-valued. Since we now support stretch-blit, we want the ability to animate the scaling factor smoothly. If the destination rectangle size is rounded to an integer, you won't get smooth animation.

New Procedures and Functions

Added to OpenGL 1.3 and OpenGL ES 1.0:

```
void DrawTex{sifx}OES(T X, T Y, T Z, T W, T H);
void DrawTex{sifx}vOES(T* coords);
```

Added to OpenGL ES 1.0:

```
void TexParameter{ifx}v(enum target, enum pname, T param);
```

New Types

None

New Tokens

Accepted by the <pname> parameter of TexParameter() and GetTexParameter():

```
TEXTURE_CROP_RECT_OES        0x8B9D
```

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation):

None

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization):

In Table 3.19: Texture parameters and their values, p. 133, add this line at the end of the table:

Name	Type	Legal Values
TEXTURE_CROP_RECT_OES	4 integers	any value

In section 3.8.4, Texture Parameters, after paragraph 3 (page 132) insert new paragraph:

The texture parameter TEXTURE_CROP_RECT_OES controls the operation of DrawTex{sifx}[v]OES(), as described in section 5.7. It has no effect on the rasterization of other primitives.

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer):

None

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions):

In Chapter 5, paragraph one, replace the last two words ("and hints.") with the words "hints, and texture rectangle drawing."

After section 5.6, p. 196, insert:

5.7 Texture Rectangle Drawing

OpenGL supports drawing sub-regions of a texture to rectangular regions of the screen using the texturing pipeline. Source region size and content are determined by the texture crop rectangle(s) of the enabled texture(s) (see section 3.8.14).

The functions

```
void DrawTex{sifx}OES(T Xs, T Ys, T Zs, T Ws, T Hs);
void DrawTex{sifx}vOES(T *coords);
```

draw a texture rectangle to the screen. Xs, Ys, and Zs specify the position of the affected screen rectangle.

Xs and Ys are given directly in window (viewport) coordinates. Zs is mapped to window depth Zw as follows:

```
             { n,                 if z <= 0
        Zw = { f,                 if z >= 1
             { n + z * (f - n),   otherwise
```

where <n> and <f> are the near and far values of DEPTH_RANGE. Ws and Hs specify the width and height of the affected screen rectangle in pixels. These values may be positive or negative; however, if either (Ws <= 0) or (Hs <= 0), the INVALID_VALUE error is generated.

Calling one of the DrawTex functions generates a fragment for each pixel that overlaps the screen rectangle bounded by (Xs, Ys) and (Xs + Ws), (Ys + Hs). For each generated fragment, the depth is given by Zw as defined above, and the color by the current color.

If EXT_fog_coord is supported, and FOG_COORDINATE_SOURCE_EXT is set to FOG_COORINATE_EXT, then the fragment distance for fog purposes is set to CURRENT_FOG_COORDINATE. Otherwise, the fragment distance for fog purposes is set to 0.

Texture coordinates for each texture unit are computed as follows:

Let X and Y be the screen x and y coordinates of each sample point associated with the fragment. Let Wt and Ht be the width and height in texels of the texture currently bound to the texture unit. (If the texture is a mipmap, let Wt and Ht be the dimensions of the level specified by TEXTURE_BASE_LEVEL.) Let Ucr, Vcr, Wcr and Hcr be (respectively) the four integers that make up the texture crop rectangle parameter for the currently bound texture. The fragment texture coordinates (s, t, r, q) are given by

```
s = (Ucr + (X - Xs)*(Wcr/Ws)) / Wt
t = (Vcr + (Y - Ys)*(Hcr/Hs)) / Ht
r = 0
q = 1
```

In the specific case where X, Y, Xs and Ys are all integers, Wcr/Ws and Hcr/Hs are both equal to one, the base level is used for the texture read, and fragments are sampled at pixel centers, implementations are required to ensure that the resulting u, v texture indices are also integers. This results in a one-to-one mapping of texels to fragments.

Note that Wcr and/or Hcr can be negative. The formulas given above for s and t still apply in this case. The result is that if Wcr is negative, the source rectangle for DrawTex operations lies to the left of the reference point (Ucr, Vcr) rather than to the right of it, and appears right-to-left reversed on the screen after a call to DrawTex. Similarly, if Hcr is negative, the source rectangle lies below the reference point (Ucr, Vcr) rather than above it, and appears upside-down on the screen.

Note also that s, t, r, and q are computed for each fragment as part of DrawTex rendering. This implies that the texture matrix is ignored and has no effect on the rendered result.

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests):

None

Additions to Appendix A of the OpenGL 1.3 Specification (Invariance):

None

Additions to the AGL/GLX/WGL Specifications:

None

Additions to Chapter 2 of the OpenGL ES 1.0 Specification (OpenGL Operation):

None

Additions to Chapter 3 of the OpenGL ES 1.0 Specification (Rasterization):

After the fourth paragraph of section 3.8, Texturing, p. 17, insert a new paragraph:

DrawTexOES is supported.

In the (unnamed) table of supported texture functions, p. 19, delete the entry for TexParameter{i[v] fv}(), and replace the entry for TexParameterf() with the following:

OpenGL 1.3	Common	Common-Lite

TexParameter{if}[v](enum target, enum param, T param)		
target = TEXTURE_2D, pname = TEXTURE_CROP_RECT_OES	(check)	(check)
target = TEXTURE_1D, TEXTURE_3D, TEXTURE_CUBE_MAP	-	-
pname = TEXTURE_MIN_FILTER, TEXTURE_MAG_FILTER	(check)	(check)
pname = TEXTURE_WRAP_S, TEXTURE_WRAP_T	(check)	(check)
pname = TEXTURE_BORDER_COLOR	-	-
pname = TEXTURE_MIN_LOD, TEXTURE_MAX_LOD	-	-
pname = TEXTURE_BASE_LEVEL, TEXTURE_MAX_LEVEL	-	-
pname = TEXTURE_WRAP_R	-	-
pname = TEXTURE_PRIORITY	-	-

In the same table, modify the entry for GetTexParameter{if}v() to read as follows:

OpenGL 1.3	Common	Common-Lite
GetTexParameter{if}v(enum target, enum param, T *params)	(check)	(dagger)

Additions to Chapter 4 of the OpenGL ES 1.0 Specification (Per-Fragment Operations and the Frame Buffer):

None

Additions to Chapter 5 of the OpenGL ES 1.0 Specification (Special Functions):

None

Additions to Chapter 6 of the OpenGL ES 1.0 Specification (State and State Requests):

At the end of table 6.15, Texture Objects (cont.), p. 36, insert a new entry:

State	Exposed	Queriable
TEXTURE_CROP_RECT_OES	(check)	(check)

Replace the fourth paragraph of Chapter 7, Core Additions and Extensions, p. 46, with the following:

The Common and Common-Lite profiles add subsets of the OES_byte_coordinates, OES_fixed_point, and OES_single_precision ES-specific extensions as core additions; OES_readFormat and OES_compressed_paletted_texture as required profile extensions; and OES_query_matrix and OES_draw_texture as optional profile extensions.

Additions to Chapter 7 of the OpenGL ES 1.0 Specification (Core Additions and Extensions):

At the end of Table 7.1: OES Extension Disposition, add a new entry:

Extension Name	Common	Common-Lite
OES_draw_texture	optional extension	optional extension

After section 7.6, Query Matrix, insert

7.7 Draw Texture

The optional OES_draw_texture extension allows rectangular subregions of a texture to be written to the screen using the fragment pipeline. Texture coordinates are generated for each fragment in the destination rectangle, such that texels in the source texture are mapped linearly to pixels on the screen.

GLX Protocol

None

Errors

None

Dependencies on OES_fixed_point

The DrawTex{sifx}[v]() function makes use of the ’x’ suffix and (in that form) accepts parameters of type fixed, as defined in OES_fixed_point.

Dependencies on EXT_fog_coord

EXT_fog_coord affects the distance that is used in the fog equations for fragments generated by DrawTex{sifx}[v](). If EXT_fog_coord is not supported, the fog distance for each fragment is set to zero. If EXT_fog_coord is supported, the fog distance depends on the value of FOG_COORDINATE_SOURCE_EXT. If the latter is set to

FRAGMENT_DEPTH_EXT, the fog distance is again set to zero. If FOG_COORDINATE_SOURCE_EXT is set to FOG_COORDINATE_EXT, the distance is set to CURRENT_FOG_COORDINATE.

New State

(table 6.16, Texture Objects (cont.), p. 224):

Get Value	Type	Get Command	Initial Value	Description	Sec	Attribute
TEXTURE_CROP_RECT	4xZ	GetTexParameteriv	0,0,0,0	texture crop rectangle	5.7	texture

New Implementation Dependent State

None.

Revision History

July 21, 2004 (v0.96)
- Modified to say that if Ws or Hs < 0 then an INVALID_VALUE error is generated

July 16, 2004 (v0.95)
- Corrected a bug in the text description of DrawTex with negative crop rectangle width or height. Thanks to Petri Kero for the catch.

July 14, 2004 (v0.9)
- added a Zs parameter to the destination rectangle location specification. This allows applications to control the depth coordinate of fragments generated by DrawTex().
- Removed DOS-mode carriage returns.

June 29, 2004 (v0.8)
- Corrected dependencies to comply with ARB recommended practice for extensions.
- Restructured the "Additions to the OpenGL ES 1.0 Spec" sections to separate changes by chapter, following ARB recommended practice for OpenGL and GLX specifications.
- Modified TexParameter usage to be consistent with OpenGL ES 1.1.
- Added a dependency on EXT_fog_coord.
- Inserted enumerant value.

June 16, 2004 (v0.7)
- Modified to make texture crop rectangle part of

```
          texture state (set by TexParameter) rather than by
          an ad hoc function (TexCropRectOES).
        - Modified to provide stretch-blit functionality.

    May   28, 2004 (v0.6)
        - Formalized changes to 1.3 and ES 1.0 specs.
          Modified to take screen coordinate arguments
          rather than using the current raster position.

    May   19, 2004 (v0.5)
        - Simplified to support only one-to-one source
          blit.  Sprite functionality was moved to a
          separate proposal.

    May    4, 2004 (v0.4)
        - Rewrote to use explicit source and destination
          rectangles instead of overloading PixelZoom.
          Made current raster rectangle explicit and
          provided both screen space and object space
          ways to define it.

    April 13, 2004
- Initial version (v0.3)
```