OpenGL$^{®}$ ES
Common/Common-Lite Profile Specification

Version 1.0.02 (Annotated)

Editor: David Blythe

Copyright (c) 2002-2004 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 **2**
- 2.1 OpenGL Fundamentals 2
  - 2.1.1 Fixed-Point Computation 3
- 2.2 GL State 3
- 2.3 GL Command Syntax 3
- 2.4 Basic GL Operation 4
- 2.5 GL Errors 4
- 2.6 Begin/End Paradigm 4
- 2.7 Vertex Specification 5
- 2.8 Vertex Arrays 6
- 2.9 Rectangles 7
- 2.10 Coordinate Transformations 8
- 2.11 Clipping 9
- 2.12 Current Raster Position 9
- 2.13 Colors and Coloring 10

3 Rasterization **12**
- 3.1 Invariance 12
- 3.2 Antialiasing 12
- 3.3 Points 12
- 3.4 Line Segments 12
- 3.5 Polygons 13
- 3.6 Pixel Rectangles 14
- 3.7 Bitmaps 16
- 3.8 Texturing 16
- 3.9 Fog 21

4 Per-Fragment Operations and the Framebuffer **22**
- 4.1 Per-Fragment Operations 22
- 4.2 Whole Framebuffer Operations 23
- 4.3 Drawing, Reading, and Copying Pixels 24

5 Special Functions 25

5.1 Evaluators 25
5.2 Selection 25
5.3 Feedback 26
5.4 Display Lists 26
5.5 Flush and Finish 26
5.6 Hints 27

6 State and State Requests 28

6.1 Querying GL State 28
6.2 State Tables 30

7 Core Additions and Extensions 46

7.1 Byte Coordinates 46
7.2 Fixed Point 47
7.3 Single-precision Commands 48
7.4 Compressed Paletted Texture 48
7.5 Read Format 49
7.6 Query Matrix 49

8 Packaging 50

8.1 Header Files 50
8.2 Libraries 50

A Acknowledgements 51

B OES Extension Specifications 52

B.1 OES_byte_coordinates 52
B.2 OES_fixed_point 55
B.3 OES_single_precision 64
B.4 OES_read_format 69
B.5 OES_query_matrix 73
B.6 OES_compressed_paletted_texture 76

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 1.3 specification.

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 1.3 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 ♢ indicates that the double-precision form of the command is replaced with its single-precision variant from the `OES_single_precision` extension. The symbol ∗ indicates that an enumerant is part of a new ES-specific extension. The superscript ‡ indicates that the command is supported subject to additional constraints described in the section body containing the table.

> ■ 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. ❑

---

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

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 1.3 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 1.3 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 1.3 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 1.3. ❑

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 1.3 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. 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 1.3. The major differences from the OpenGL 1.3 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 1.3	**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 1.3, it may be necessary to call **GetError** multiple times to retrieve error state from all parts of the pipeline.

OpenGL 1.3	**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 primitives: `POINTS`, `LINES`, `LINE_STRIP`, `LINE_LOOP`, `TRIANGLES`, `TRIANGLE_STRIP`, and `TRIANGLE_FAN` are supported; the primitives: `QUADS`, `QUAD_STRIP`, and `POLYGON` are not supported.

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

OpenGL 1.3	**Common**	**Common-Lite**
Begin`(enum mode)`	–	–
End`(void)`	–	–
EdgeFlag[v]`(T flag)`	–	–

■ The Begin/End paradigm, while convenient and efficient, 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**.

Multitexture texture coordinates are supported, though only a single texture unit needs to be supported.

OpenGL 1.3	**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)`	–	–
Color4f`(float components)`	✓	†
Color{**34**}{**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 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 1.3	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 = INT, DOUBLE`	–	–
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`	–	–

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`	–	–
DisableClientState`(enum cap)`		
`cap = TEXTURE_COORD_ARRAY,COLOR_ARRAY`	✓	✓
`cap = NORMAL_ARRAY,VERTEX_ARRAY`	✓	✓
`cap = EDGE_FLAG_ARRAY, INDEX_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 1.3 specification defines the initial type for each of the arrays to be `FLOAT`. Since the array lengths are zero and the types cannot be queried by the application, this initial state is essentially invisible to an application and minimizes a potential inconsistency with the removal of floating-point data types from the Common-Lite profile.

Multitexture is included on the assumption that it will be supported by hardware accelerators in the near term and it would be better to include it from the start rather than adding it during the first revision. ❑

2.9 Rectangles

The commands for directly specifying rectangles are not supported.

OpenGL 1.3	**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. ❑

2.10 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 1.3	**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]`)	✓	†
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. ❑

2.11 Clipping

Clipping against the viewing frustum is supported; however, separate user-specified clipping planes are not supported.

OpenGL 1.3	**Common**	**Common-Lite**
ClipPlane(`enum plane, const double *equation`)	–	–
GetClipPlane(`enum plane, double *equation`)	–	–
Enable/Disable(`CLIP_PLANE{0-5}`)	–	–

■ User-specified clipping planes are used predominately in engineering and scientific applications. It would be useful to have at least one clipping plane for some ”portal-culling” algorithms, but there hasn’t been a strong enough case made for keeping them. ❑

2.12 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 1.3	**Common**	**Common-Lite**
RasterPos{**2,3,4**}{**sifd**}**[v]**(`T coords`)	–	–

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

2.13 Colors and Coloring

The OpenGL 1.3 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 1.3 default infinite viewer and single color computational models are supported. Smooth and flat shading are fully supported for all primitives.

OpenGL 1.3	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)`	–	–
GetMaterial{if}[v]`(enum face, enum pname, T *params)`	–	–
Lightf[v]`(enum light, enum pname, T param)`	✓	†
Lighti[v]`(enum light, enum pname, T param)`	–	–
GetLight{if}[v]`(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 1.3	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.

OpenGL 1.3	Common	Common-Lite
PointSize(`float size`)	✓	†
Enable/Disable(`POINT_SMOOTH`)	✓	✓

■ See below. ❑

3.4 Line Segments

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

OpenGL 1.3	Common	Common-Lite
LineWidth(`float width`)	✓	†
Enable/Disable(`LINE_SMOOTH`)	✓	✓
LineStipple(`int factor, ushort pattern`)	–	–
Enable/Disable(`LINE_STIPPLE`)	–	–

■ 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 1.3 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 1.3	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 1.3	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)	–	–

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)	–	–
GetConvolutionParameterfv(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 1.3 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 **Draw-**

Pixels 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 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 1.3	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

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. Texture borders are not supported (the **border** parameter must be zero, and an `INVALID_VALUE` error results if it is non-zero).

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 1.3	Common	Common-Lite
`UNSIGNED_BYTE`	✓	✓
`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

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.

Wrap modes `REPEAT` and `CLAMP_TO_EDGE` are supported, but not `CLAMP` and `CLAMP_TO_BORDER`. Texture priorities, LOD clamps, and explicit base and maximum level specification are not supported. The remaining OpenGL 1.3 texture parameters are supported including all filtering modes. Texture objects are supported, but proxy textures are not supported. Multitexture is supported, but the `COMBINE` texture environment mode is not.

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

OpenGL 1.3	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)		
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	–	–
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)	–	–

TexParameterf(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_PRIORITY	–	–
TexParameter{**i[v] fv**}(enum target, enum pname, T param)	–	–
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	✓	✓
cap = TEXTURE_1D,TEXTURE_3D,TEXTURE_CUBE_MAP	–	–
TexEnvf[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 = SOURCE{012}_RGB,SOURCE{012}_ALPHA	–	–
pname = RGB_SCALE,ALPHA_SCALE	–	–
TexEnvi[v](enum target, enum pname, T param)	–	–
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.

All OpenGL 1.3 texture environments except for the combine mode are supported. Combine is not supported as it creates a substantial implementation burden and is expected to be replaced with pixel shaders in some future version. Compressed textures are important for reducing space and bandwidth requirements. The OpenGL 1.3 compression infrastructure is retained (for 2D textures) and a simple palette-based compression format is added as a required profile extension.

A texture is considered incomplete in OpenGL ES if the set of mipmap arrays are not specified with the same type. The check for completeness is done when a given texture is used to render geometry. ❑

3.9 Fog

Fog is fully supported except for color index related modes.

OpenGL 1.3	Common	Common-Lite
Fogf[v](`enum pname, T param`)		
pname = `FOG_MODE`,`FOG_DENSITY`,`FOG_START`,`FOG_END`,`FOG_COLOR`	✓	†
pname = `FOG_INDEX`	–	–
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 1.3 per-fragment operations are supported, except for color index related operations and the imaging subset additions (**BlendColor** and **BlendEquation**). Depth and stencil operations are supported, but an implementation is not required to include a depth or stencil buffer.

OpenGL 1.3	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`)	✓	✓
StencilMask(`uint mask`)	✓	✓
StencilOp(`enum fail, enum zfail, enum zpass`)	✓	✓
Enable/Disable(`DEPTH_TEST`)	✓	✓
DepthFunc(`enum func`)	✓	✓
DepthMask(`boolean flag`)	✓	✓
Enable/Disable(`BLEND`)	✓	✓
BlendFunc(`enum sfactor, enum dfactor`)	✓	✓

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)	✓	✓

■ 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 so they 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 1.3	Common	Common-Lite
DrawBuffer(enum mode)	–	–
IndexMask(uint mask)	–	–
ColorMask(boolean red, boolean green, boolean blue, boolean alpha)	✓	✓
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 1.3	**Common**	**Common-Lite**
ReadBuffer`(enum mode)`	–	–
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 1.3	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 1.3	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 1.3	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 1.3	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 1.3	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 1.3	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`	–	–

■ 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* state explicitly supported in the profile, such as implementation-specific constants. Commands that query non-simple dynamic state, such as **GetLight** or **GetMaterial** are not part of the profile. The simple state query command **GetIntegerv** is retained to allow querying of static state. The simple state variables are listed in Table 6.2: supported static state is shown in **boldface**. In addition to this subset, queries of the extension state: `IMPLEMENTATION_COLOR_READ_TYPE_OES` and `IMPLEMENTATION_COLOR_READ_FORMAT_OES` are supported.

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 1.3
`RENDERER`	as defined by OpenGL 1.3
`VERSION`	”`OpenGL ES-XX 1.0`” XX={`CM`,`CL`}
`EXTENSIONS`	as defined by OpenGL 1.3

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.

ACTIVE_TEXTURE	**ALIASED_LINE_WIDTH_RANGE**	**ALIASED_POINT_SIZE_RANGE**
ALPHA_BITS	ALPHA_TEST	ALPHA_TEST_FUNC
ALPHA_TEST_REF	BLEND	BLEND_DST
BLEND_SRC	**BLUE_BITS**	CLIENT_ACTIVE_TEXTURE
COLOR_ARRAY	COLOR_ARRAY_POINTER	COLOR_ARRAY_SIZE
COLOR_ARRAY_STRIDE	COLOR_ARRAY_TYPE	COLOR_CLEAR_VALUE
COLOR_LOGIC_OP	COLOR_MATERIAL	COLOR_MATERIAL_FACE
COLOR_MATERIAL_PARAMETER	COLOR_WRITEMASK	**COMPRESSED_TEXTURE_FORMATS**
CULL_FACE	CULL_FACE_MODE	CURRENT_COLOR
CURRENT_NORMAL	CURRENT_RASTER_COLOR	CURRENT_RASTER_DISTANCE
CURRENT_RASTER_INDEX	CURRENT_RASTER_POSITION	CURRENT_RASTER_POSITION_VALID
CURRENT_RASTER_TEXTURE_COORDS	CURRENT_TEXTURE_COORDS	**DEPTH_BITS**
DEPTH_CLEAR_VALUE	DEPTH_FUNC	DEPTH_RANGE
DEPTH_TEST	DEPTH_WRITEMASK	DITHER
DOUBLEBUFFER	DRAW_BUFFER	FOG
FOG_COLOR	FOG_DENSITY	FOG_END
FOG_HINT	FOG_MODE	FOG_START
FRONT_FACE	**GREEN_BITS**	LIGHT0
LIGHT1	LIGHT2	LIGHT3
LIGHT4	LIGHT5	LIGHT6
LIGHT7	LIGHTING	LIGHT_MODEL_AMBIENT
LIGHT_MODEL_COLOR_CONTROL	LIGHT_MODEL_LOCAL_VIEWER	LIGHT_MODEL_TWO_SIDE
LINE_SMOOTH	LINE_SMOOTH_HINT	LINE_WIDTH
LINE_WIDTH_GRANULARITY	LINE_WIDTH_RANGE	LOGIC_OP_MODE
MATRIX_MODE	**MAX_ELEMENTS_INDICES**	**MAX_ELEMENTS_VERTICES**
MAX_LIGHTS	**MAX_MODELVIEW_STACK_DEPTH**	**MAX_PROJECTION_STACK_DEPTH**
MAX_TEXTURE_SIZE	**MAX_TEXTURE_STACK_DEPTH**	**MAX_TEXTURE_UNITS**
MAX_VIEWPORT_DIMS	MODELVIEW_MATRIX	MODELVIEW_STACK_DEPTH
NORMAL_ARRAY	NORMAL_ARRAY_POINTER	NORMAL_ARRAY_STRIDE
NORMAL_ARRAY_TYPE	NORMALIZE	**NUM_COMPRESSED_TEXTURE_FORMATS**
PACK_ALIGNMENT	PERSPECTIVE_CORRECTION_HINT	POINT_SIZE
POINT_SIZE_GRANULARITY	POINT_SIZE_RANGE	POINT_SMOOTH
POINT_SMOOTH_HINT	POLYGON_OFFSET_FACTOR	POLYGON_OFFSET_FILL
POLYGON_OFFSET_LINE	POLYGON_OFFSET_POINT	POLYGON_OFFSET_UNITS
POLYGON_SMOOTH	POLYGON_SMOOTH_HINT	PROJECTION_MATRIX
PROJECTION_STACK_DEPTH	**RED_BITS**	RESCALE_NORMAL
RGBA_MODE	SCISSOR_BOX	SCISSOR_TEST
SHADE_MODEL	SMOOTH_LINE_WIDTH_GRANULARITY	**SMOOTH_LINE_WIDTH_RANGE**
SMOOTH_POINT_SIZE_GRANULARITY	**SMOOTH_POINT_SIZE_RANGE**	**STENCIL_BITS**
STENCIL_CLEAR_VALUE	STENCIL_FAIL	STENCIL_FUNC
STENCIL_PASS_DEPTH_FAIL	STENCIL_PASS_DEPTH_PASS	STENCIL_REF
STENCIL_TEST	STENCIL_VALUE_MASK	STENCIL_WRITEMASK
STEREO	**SUBPIXEL_BITS**	TEXTURE_1D
TEXTURE_2D	TEXTURE_3D	TEXTURE_BINDING_1D
TEXTURE_BINDING_2D	TEXTURE_BINDING_3D	TEXTURE_COORD_ARRAY
TEXTURE_COORD_ARRAY_POINTER	TEXTURE_COORD_ARRAY_SIZE	TEXTURE_COORD_ARRAY_STRIDE
TEXTURE_COORD_ARRAY_TYPE	TEXTURE_MATRIX	TEXTURE_STACK_DEPTH
UNPACK_ALIGNMENT	VERTEX_ARRAY	VERTEX_ARRAY_POINTER
VERTEX_ARRAY_SIZE	VERTEX_ARRAY_STRIDE	VERTEX_ARRAY_TYPE
VIEWPORT		

Table 6.2: Static and Dynamic State: Queriable state in boldface

OpenGL 1.3	**Common**	**Common-Lite**
GetBooleanv`(enum pname, boolean *params)`	–	–
GetDoublev`(enum pname, double *params)`	–	–
GetFloatv`(enum pname, float *params)`	–	–
GetIntegerv`(enum pname, int *params)`	✓	✓
IsEnabled`(enum cap)`	–	–
GetString`(enum name)`	✓	✓
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 and use and therefore not supported. Generally speaking state queries are discouraged as they are often detrimental to performance. Rather than trying to partition different types of dynamic state that can be queried, tops of matrix stacks for example, no dynamic state queries are supported and applications must shadow state changes rather than querying the pipeline. This makes things difficult for layered libraries, but there hasn't been enough justification to retain dynamic state queries or attribute pushing and popping.

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. State appearing in *italic* indicates unnamed state. All state has initial values identical to those specified in OpenGL 1.3.

State	**Exposed**	**Queriable**
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**
`CURRENT_COLOR`	✓	–
`CURRENT_INDEX`	–	–
`CURRENT_TEXTURE_COORDS`	✓	–
`CURRENT_NORMAL`	✓	–
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
CLIENT_ACTIVE_TEXTURE	✓	–
VERTEX_ARRAY	✓	–
VERTEX_ARRAY_SIZE	✓	–
VERTEX_ARRAY_STRIDE	✓	–
VERTEX_ARRAY_TYPE	✓	–
VERTEX_ARRAY_POINTER	✓	–
NORMAL_ARRAY	✓	–
NORMAL_ARRAY_STRIDE	✓	–
NORMAL_ARRAY_TYPE	✓	–
NORMAL_ARRAY_POINTER	✓	–
COLOR_ARRAY	✓	–
COLOR_ARRAY_SIZE	✓	–
COLOR_ARRAY_STRIDE	✓	–
COLOR_ARRAY_TYPE	✓	–
COLOR_ARRAY_POINTER	✓	–
INDEX_ARRAY	–	–
INDEX_ARRAY_STRIDE	–	–
INDEX_ARRAY_TYPE	–	–
INDEX_ARRAY_POINTER	–	–

Table 6.6: Vertex Array Data

State	Exposed	Queriable
TEXTURE_COORD_ARRAY	✓	–
TEXTURE_COORD_ARRAY_SIZE	✓	–
TEXTURE_COORD_ARRAY_STRIDE	✓	–
TEXTURE_COORD_ARRAY_TYPE	✓	–
TEXTURE_COORD_ARRAY_POINTER	✓	–
EDGE_FLAG_ARRAY	–	–
EDGE_FLAG_ARRAY_STRIDE	–	–
EDGE_FLAG_ARRAY_POINTER	–	–

Table 6.7: Vertex Array Data (cont.)

State	**Exposed**	**Queriable**
`COLOR_MATRIX`	–	–
`MODEL_VIEW_MATRIX`	✓	–
`PROJECTION_MATRIX`	✓	–
`TEXTURE_MATRIX`	✓	–
`VIEWPORT`	✓	–
`DEPTH_RANGE`	✓	–
`COLOR_MATRIX_STACK_DEPTH`	–	–
`MODELVIEW_STACK_DEPTH`	✓	–
`PROJECTION_STACK_DEPTH`	✓	–
`TEXTURE_STACK_DEPTH`	✓	–
`MATRIX_MODE`	✓	–
`NORMALIZE`	✓	–
`RESCALE_NORMAL`	✓	–
`CLIP_PLANE{0-5}`	–	–

Table 6.8: Transformation State

State	**Exposed**	**Queriable**
`FOG_COLOR`	✓	–
`FOG_INDEX`	–	–
`FOG_DENSITY`	✓	–
`FOG_START`	✓	–
`FOG_END`	✓	–
`FOG_MODE`	✓	–
`FOG`	✓	–
`SHADE_MODEL`	✓	–

Table 6.9: Coloring

State	Exposed	Queriable
`LIGHTING`	✓	–
`COLOR_MATERIAL`	✓	–
`COLOR_MATERIAL_PARAMETER`	–	–
`COLOR_MATERIAL_FACE`	–	–
`AMBIENT (material)`	✓	–
`DIFFUSE (material)`	✓	–
`SPECULAR (material)`	✓	–
`EMISSION (material)`	✓	–
`SHININESS (material)`	✓	–
`LIGHT_MODEL_AMBIENT`	✓	–
`LIGHT_MODEL_LOCAL_VIEWER`	–	–
`LIGHT_MODEL_TWO_SIDE`	✓	–
`LIGHT_MODEL_COLOR_CONTROL`	–	–

Table 6.10: Lighting

State	Exposed	Queriable
AMBIENT (light$_i$)	✓	–
DIFFUSE (light$_i$)	✓	–
SPECULAR (light$_i$)	✓	–
EMISSION (light$_i$)	✓	–
`CONSTANT_ATTENUATION`	✓	–
`LINEAR_ATTENUATION`	✓	–
`QUADRATIC_ATTENUATION`	✓	–
`SPOT_DIRECTION`	✓	–
`SPOT_EXPONENT`	✓	–
`SPOT_CUTOFF`	✓	–
`LIGHT{0-7}`	✓	–
`COLOR_INDEXES`	–	–

Table 6.11: Lighting (cont.)

State	**Exposed**	**Queriable**
`POINT_SIZE`	✓	–
`POINT_SMOOTH`	✓	–
`LINE_WIDTH`	✓	–
`LINE_SMOOTH`	✓	–
`LINE_STIPPLE_PATTERN`	–	–
`LINE_STIPPLE_REPEAT`	–	–
`LINE_STIPPLE`	–	–
`CULL_FACE`	✓	–
`CULL_FACE_MODE`	✓	–
`FRONT_FACE`	✓	–
`POLYGON_SMOOTH`	–	–
`POLYGON_MODE`	–	–
`POLYGON_OFFSET_FACTOR`	✓	–
`POLYGON_OFFSET_UNITS`	✓	–
`POLYGON_OFFSET_POINT`	–	–
`POLYGON_OFFSET_LINE`	–	–
`POLYGON_OFFSET_FILL`	✓	–
`POLYGON_STIPPLE`	–	–

Table 6.12: Rasterization

State	**Exposed**	**Queriable**
`MULTISAMPLE`	✓	–
`SAMPLE_ALPHA_TO_COVERAGE`	✓	–
`SAMPLE_ALPHA_TO_ONE`	✓	–
`SAMPLE_COVERAGE`	✓	–
`SAMPLE_COVERAGE_VALUE`	✓	–
`SAMPLE_COVERAGE_INVERT`	✓	–

Table 6.13: Multisampling

State	Exposed	Queriable
`TEXTURE_1D`	–	–
`TEXTURE_2D`	✓	–
`TEXTURE_3D`	–	–
`TEXTURE_CUBE_MAP`	–	–
`TEXTURE_BINDING_1D`	–	–
`TEXTURE_BINDING_2D`	✓	–
`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`	–	–

Table 6.14: Texture Objects

State	Exposed	Queriable
`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`	–	–

Table 6.15: Texture Objects (cont.)

State	**Exposed**	**Queriable**
TEXTURE_COMPRESSED	✓	–
TEXTURE_COMPRESSED_IMAGE_SIZE	✓	–
TEXTURE_BORDER_COLOR	–	–
TEXTURE_MIN_FILTER	✓	–
TEXTURE_MAG_FILTER	✓	–
TEXTURE_WRAP_S	✓	–
TEXTURE_WRAP_T	✓	–
TEXTURE_WRAP_R	–	–
TEXTURE_PRIORITY	–	–
TEXTURE_RESIDENT	–	–
TEXTURE_MIN_LOD	✓	–
TEXTURE_MAX_LOD	✓	–
TEXTURE_BASE_LEVEL	✓	–
TEXTURE_MAX_LEVEL	✓	–

Table 6.16: Texture Objects (cont.)

State	**Exposed**	**Queriable**
ACTIVE_TEXTURE	✓	–
TEXTURE_ENV_MODE	✓	–
TEXTURE_ENV_COLOR	✓	–
TEXTURE_GEN_{STRQ}	–	–
EYE_PLANE	–	–
OBJECT_PLANE	–	–
TEXTURE_GEN_MODE	–	–
COMBINE_RGB	–	–
COMBINE_ALPHA	–	–
SOURCE{012}_RGB	–	–
SOURCE{012}_ALPHA	–	–
OPERAND{012}_RGB	–	–
OPERAND{012}_ALPHA	–	–
RGB_SCALE	–	–
ALPHA_ALPHA	–	–

Table 6.17: Texture Environment and Generation

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

Table 6.18: Pixel Operations

State	**Exposed**	**Queriable**
`DRAW_BUFFER`	–	–
`INDEX_WRITEMASK`	–	–
`COLOR_WRITEMASK`	✓	–
`DEPTH_WRITEMASK`	✓	–
`STENCIL_WRITEMASK`	✓	–
`COLOR_CLEAR_VALUE`	✓	–
`INDEX_CLEAR_VALUE`	–	–
`DEPTH_CLEAR_VALUE`	✓	–
`STENCIL_CLEAR_VALUE`	✓	–
`ACCUM_CLEAR_VALUE`	–	–

Table 6.19: Framebuffer Control

State	Exposed	Queriable
`UNPACK_SWAP_BYTES`	–	–
`UNPACK_LSB_FIRST`	–	–
`UNPACK_IMAGE_HEIGHT`	–	–
`UNPACK_SKIP_IMAGES`	–	–
`UNPACK_ROW_LENGTH`	–	–
`UNPACK_SKIP_ROWS`	–	–
`UNPACK_SKIP_PIXELS`	–	–
`UNPACK_ALIGNMENT`	✓	–
`PACK_SWAP_BYTES`	–	–
`PACK_LSB_FIRST`	–	–
`PACK_IMAGE_HEIGHT`	–	–
`PACK_SKIP_IMAGES`	–	–
`PACK_ROW_LENGTH`	–	–
`PACK_SKIP_ROWS`	–	–
`PACK_SKIP_PIXELS`	–	–
`PACK_ALIGNMENT`	✓	–
`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 6.20: Pixels

State	**Exposed**	**Queriable**
`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 6.21: Pixels (cont.)

State	**Exposed**	**Queriable**
`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 6.22: Pixels (cont.)

State	**Exposed**	**Queriable**
`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 6.23: Pixels (cont.)

State	**Exposed**	**Queriable**
`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 6.24: Pixels (cont.)

State	**Exposed**	**Queriable**
`ORDER`	–	–
`COEFF`	–	–
`DOMAIN`	–	–
`MAP1_`*x*	–	–
`MAP2_`*x*	–	–
`MAP1_GRID_DOMAIN`	–	–
`MAP2_GRID_DOMAIN`	–	–
`MAP1_GRID_SEGMENTS`	–	–
`MAP2_GRID_SEGMENTS`	–	–
`AUTO_NORMAL`	–	–

Table 6.25: Evaluators

State	**Exposed**	**Queriable**
`PERSPECTIVE_CORRECTION_HINT`	✓	✓
`POINT_SMOOTH_HINT`	✓	✓
`LINE_SMOOTH_HINT`	✓	✓
`POLYGON_SMOOTH_HINT`	–	–
`FOG_HINT`	✓	✓
`TEXTURE_COMPRESSION_HINT`	✓	✓

Table 6.26: Hints

State	**Exposed**	**Queriable**
`MAX_CLIP_PLANES`	–	–
`MAX_COLOR_MATRIX_STACK_DEPTH`	–	–
`MAX_MODELVIEW_STACK_DEPTH`	✓	✓
`MAX_PROJECTION_STACK_DEPTH`	✓	✓
`MAX_TEXTURE_STACK_DEPTH`	✓	✓
`SUBPIXEL_BITS`	✓	✓
`MAX_3D_TEXTURE_SIZE`	–	–
`MAX_TEXTURE_SIZE`	✓	✓
`MAX_CUBE_MAP_TEXTURE_SIZE`	–	–
`MAX_PIXEL_MAP_TABLE`	–	–
`MAX_NAME_STACK_DEPTH`	–	–
`MAX_LIST_NESTING`	–	–
`MAX_EVAL_ORDER`	–	–
`MAX_VIEWPORT_DIMS`	✓	✓

Table 6.27: Implementation Dependent Values

State	**Exposed**	**Queriable**
`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`	✓	✓
`SMOOTH_POINT_SIZE_RANGE`	✓	✓
`SMOOTH_POINT_SIZE_GRANULARITY`	✓	–
`ALIASED_LINE_WIDTH_RANGE`	✓	✓
`SMOOTH_LINE_WIDTH_RANGE`	✓	✓
`SMOOTH_LINE_WIDTH_GRANULARITY`	✓	–

Table 6.28: Implementation Dependent Values (cont.)

State	**Exposed**	**Queriable**
`MAX_CONVOLUTION_WIDTH`	–	–
`MAX_CONVOLUTION_HEIGHT`	–	–
`MAX_ELEMENTS_INDICES`	✓	–
`MAX_ELEMENTS_VERTICES`	✓	–
`MAX_TEXTURE_UNITS`	✓	✓
`SAMPLE_BUFFERS`	✓	✓
`SAMPLES`	✓	✓
`COMPRESSED_TEXTURE_FORMATS`	✓	✓
`NUM_COMPRESSED_TEXTURE_FORMATS`	✓	✓

Table 6.29: Implementation Dependent Values (cont.)

State	**Exposed**	**Queriable**
`RED_BITS`	✓	✓
`GREEN_BITS`	✓	✓
`BLUE_BITS`	✓	✓
`ALPHA_BITS`	✓	✓
`INDEX_BITS`	–	–
`DEPTH_BITS`	✓	✓
`STENCIL_BITS`	✓	✓
`ACCUM_BITS`	–	–

Table 6.30: Implementation Dependent Pixel Depths

State	**Exposed**	**Queriable**
`LIST_INDEX`	–	–
`LIST_MODE`	–	–
Server attribute stack	–	–
`ATTRIB_STACK_DEPTH`	–	–
Client attribute stack	–	–
`CLIENT_ATRIB_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)	✓	✓
Corresponding error flags	✓	✓

Table 6.31: Miscellaneous

State	Exposed	Queriable
`IMPLEMENTATION_COLOR_READ_TYPE_OES`	✓	✓
`IMPLEMENTATION_COLOR_READ_FORMAT_OES`	✓	✓

Table 6.32: 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 are 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`, and `OES_single_precision` ES-specific extensions as core additions; `OES_read_format` and `OES_compressed_paletted_texture` as required profile extensions; and `OES_query_matrix` as an optional profile extension.

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.

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_read_format`	required extension	required extension
`OES_compressed_paletted_texture`	required extension	required extension
`OES_query_matrix`	optional extension	optional extension

Table 7.1: OES Extension Disposition

7.2 Fixed Point

The `OES_fixed_point` extension defines an integer fixed-point data type for use as vertex attributes and command parameters. The extension specification includes commands that parallel all OpenGL 1.3 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 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)`

Materialx[v]`(enum face, enum pname, T param)`
Lightx[v]`(enum light, enum pname, T param)`
LightModelx[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)`

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)`

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 Query Matrix

The optional `OES_query_matrix` extension allows the current modelview, texture, or projection matrix to be retrieved to assist with diagnostics and debugging during application development. The command allows retrieval of separate mantissa and exponent values so that an implementation of a fixed-point profile with internal dynamic range greater than 16.16 can return full range results.

Chapter 8

Packaging

8.1 Header Files

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

8.2 Libraries

Each profile defines a distinct link-library. The library name includes the profile name as `libGLES_nn.z` where `nn` is either `CM` or `CL` and `.z` is a platform-specific library suffix (i.e., `.a`, `.so`, `.lib`, etc.). The symbols for the platform-specific embedding library are also included in the link-library. Availability of static and dynamic function bindings is platform dependent. Rules regarding the export of bindings for core additions, required profile extensions, and optional platform extensions are described in Chapter 7.

Appendix A

Acknowledgements

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

Aaftab Munshi, ATI

Andy Methley, Panasonic

Carl Korobkin, 3d4W

Chris Hall, Seaweed Systems

Claude Knaus, Silicon Graphics

David Blythe, 3d4W

Ed Plowman, ARM

Graham Connor, Imagination Technologies

Harri Holopainen, Hybrid Graphics

Jacob Strom, Ericsson

Jani Vaarala, Nokia

Jon Leech, Silicon Graphics

Justin Couch, Yumetech

Kari Pulli, Nokia

Lane Roberts, Symbian

Mark Callow, HI

Mark Tarlton, Motorola

Mike Olivarez, Motorola

Neil Trevett, 3Dlabs

Phil Huxley, Tao Group

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

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.

Version

Last Modifed Date: 12 July 2003
Author Revision: 0.7

Number

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 2ˆ16.
      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 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 +/-2ˆ-15. The maximum representable magnitude for a fixed-point number used to represent positional or normal coordinates must be at least 2ˆ15; the maximum representable magnitude for colors or texture coordinates must be at least 2ˆ10. The maximum representable magnitude for all other fixed-point values must be at least 2ˆ15. x*0 = 0*x = 0. 1*x = x*1 = x. x + 0 = 0 + x = x. 0ˆ0 = 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
```

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.

Version

Last Modifed Date: 22 July 2003
Author Revision : 0.4

Number

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 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)

B.4 OES_read_format

Name

OES_read_format

Name Strings

GL_OES_read_format

Contact

David Blythe (blythe ’at’ bluevoid.com)

Status

Ratified by the Khronos BOP, July 23, 2003.

Version

Last Modifed Date: July 8, 2003
Author Revision: 0.2

Number

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.

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 type combination that provides optimal performance 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

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

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] * 2ˆexponent[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

Affie Munshi, ATI (amunshi@ati.com)

Notice

IP Status

No known IP issues

Status

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

Version

Last Modifed Date: 21 July 2004
Author Revision: 0.5

Number

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 "level" 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.

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   (Affie 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   (Affie Munshi)
- Fixed PALETTE_8xxx drawing