OpenGL® ES
Safety Critical Profile Specification

Version 1.0 (Annotated)

Editor: Chris Hall
Editor: Claude Knaus

Copyright © 2002-2005 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 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
2.2 GL State 3
2.3 GL Command Syntax 3
2.4 Basic GL Operation 3
2.5 GL Errors 3
2.6 Begin/End Paradigm 4
2.7 Vertex Specification 4
2.8 Vertex Arrays 5
2.9 Rectangles 6
2.10 Coordinate Transformations 7
2.11 Clipping 8
2.12 Current Raster Position 8
2.13 Colors and Coloring 9

3 Rasterization **11**
3.1 Invariance 11
3.2 Antialiasing 11
3.3 Points 11
3.4 Line Segments 11
3.5 Polygons 12
3.6 Pixel Rectangles 13
3.7 Bitmaps 15
3.8 Texturing 15
3.9 Fog 20

4 Per-Fragment Operations and the Framebuffer **21**
4.1 Per-Fragment Operations 21
4.2 Whole Framebuffer Operations 23
4.3 Drawing, Reading, and Copying Pixels 23

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 27
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 **44**
7.1 Single-precision Commands 45
7.2 Paletted Textures 45
7.3 Shared Texture Palette 46

8 Packaging **47**
8.1 Header Files 47
8.2 Libraries 47

A Acknowledgements **48**

B OES Extension Specifications **49**
B.1 OES_single_precision 49
B.2 EXT_paletted_texture 54
B.3 EXT_shared_texture_palette 65

Chapter 1

Overview

This document outlines the OpenGL ES Safety Critical profile. The 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 Safety Critical renderer. This document is defined relative to the OpenGL 1.3 specification.

This document specifies the OpenGL Safety Critical renderer. At this moment, no standard bindings to window systems or OS platforms exist. It is assumed that in the future, a companion document will define one or more bindings to window system/OS platform combinations analogous to the GLX, WGL, and AGL specifications. If required, an additional companion document will describe the 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 Safety Critical 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 Safety Critical profile. Several symbols are used within the tables to indicate various special cases. The symbol $\diamondsuit$ indicates that the double-precision form of the command is replaced with its single-precision variant from the `OES_single_precision` extension. The 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. ❑

Chapter 2

OpenGL Operation

The basic GL operation remains largely unchanged. A significant change in the Safety Critical profile is that the first stage of the pipeline for approximating curve and surface geometry is eliminated. The remaining pipeline stages include: display list processing, per-vertex operations and primitive assembly, pixel operations, rasterization, per-fragment operations, and whole framebuffer operations.

The Common/Common-Lite profile introduced 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. Some of these extensions are used by the Safety Critical profile as well. 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.

2.2 GL State

The Safety Critical profile retains 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 Safety Critical profile.

2.3 GL Command Syntax

Commands using the suffixes for the types: `byte`, `short`, and `ushort` are not supported. The type `double` and all double-precision commands are eliminated. The result is that the Safety Critical profile uses only the suffixes 'f', 'i' and 'ub'.

2.4 Basic GL Operation

The basic command operation remains identical to OpenGL 1.3. The major difference from the OpenGL 1.3 pipeline is that there is no polynomial function evaluation stage.

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	Safety Critical
`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	Safety Critical
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. Implementation errors (e.g. the hardware does not respond) are outside the scope of OpenGL ES and must be handled at the level of the OS binding layer. ❑

2.6 Begin/End Paradigm

The Safety Critical profile supports the concept of Begin/End and display lists. Edge flags are not supported.

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	**Safety Critical**
Begin `(enum mode)`	
`mode = POINTS,LINES,LINE_STRIP,LINE_LOOP`	✓
`mode = TRIANGLES,TRIANGLE_STRIP,TRIANGLE_FAN`	✓
`mode = QUADS,QUAD_STRIP,POLYGON`	–
End `(void)`	✓
EdgeFlag[v] `(T flag)`	–

■ The Common profile includes vertex arrays since 1.0 and vertex buffer objects since 1.1, but dropped the Begin/End paradigm including display lists to reduce implementation size. The Begin/End paradigm is supported because there are a majority of 2D Safety Critical certifiable/qualifiable applications and tools that rely on this mechanism. The Begin/End paradigm enables a melding of program and data in such a way as to allow a certifying authority to more clearly understand an application.

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 the ambiguity with respect to decomposition of these primitives into triangles, since it is entirely left to the application. `float` types are supported for all-around generality. ❑

2.7 Vertex Specification

Vertices can be specified between **Begin** and **End**, as well as with vertex arrays. Only `float`, 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 commands **Color4**, **Normal3**, and **MultiTexCoord2**.

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

OpenGL 1.3	**Safety Critical**
Vertex{23}f[v] `(T coords)`	✓
Vertex{234}{sid}[v] `(T coords)`	–

Vertex4f[v]`(T coords)`	–
Normal3f[v]`(float coords)`	✓
Normal3{bsid}[v]`(T coords)`	–
TexCoord{1234}{sifd}[v]`(T coords)`	–
MultiTexCoord2f`(enum texture, float coords)`	✓
MultiTexCoord2fv`(enum texture, float coords)`	–
MultiTexCoord2{sid}[v]`(enum texture, float coords)`	–
MultiTexCoord134{sifd}[v]`(enum texture, T coords)`	–
Color4{f fv ub}`(float components)`	✓
Color4{bsid us ui}[v]`(T components)`	–
Color3{bsifd ub us ui}[v]`(T components)`	–
Index{sifd ub}[v]`(T components)`	–

■ The Begin/End paradigm commands that have been included are based on common usage in Safety Critical applications and code-generation tools. These have been augmented with the aim of forming an orthogonal set – e.g. if a usage is available for a normal, then it is also available for coordinate data, etc.

Only 2-d texture coordinates can be specified – only 2-d textures are available, and texture matrix operations are not supported – so only 2 coordinates are useful.

The Safety Critical profile supports only the RGBA rendering model. Color index rendering is not a requirement. ❑

2.8 Vertex Arrays

Vertex arrays are supported with the same set of primitives available to Begin/End. Color index and edge flags are not supported. Both indexed and non-indexed arrays are supported, but the **InterleavedArrays**, **ArrayElement** and **DrawRangeElements** commands are not supported.

OpenGL 1.3	Safety Critical
VertexPointer`(int size, enum type, sizei stride, const void *ptr)`	
`size = 2,3 type = FLOAT`	✓
`size = 4   type = FLOAT`	–
`size = *   type = INT,SHORT,DOUBLE`	–
NormalPointer`(enum type, sizei stride, const void *ptr)`	
`type = FLOAT`	✓
`type = BYTE,UNSIGNED_BYTE,SHORT,UNSIGNED_SHORT`	–
`type = INT,UNSIGNED_INT,DOUBLE`	–
ColorPointer`(int size, enum type, sizei stride, const void *ptr)`	
`type = UNSIGNED_BYTE,FLOAT`	✓
`type = BYTE,SHORT,UNSIGNED_SHORT`	–
`type = INT,UNSIGNED_INT,DOUBLE`	–

TexCoordPointer`(int size, enum type, sizei stride, const void *ptr)`	
`size = 2 type = FLOAT`	✓
all other combinations	–
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_INT`	✓
`type = UNSIGNED_SHORT`	–
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)`	✓
DisableClientState`(enum cap)`	✓

■ The Common profile 1.0 and 1.1 support the same set of functions. However, the supported types were limited to the ones commonly used by Safety Critical applications.

Vertex arrays offer a performance improvement over Begin/End, particularly when rendering many elements of the same primitive at a time, such as in terrain rendering applications.

To further improve performance, vertex buffer objects will be a candidate for 1.1. ❑

2.9 Rectangles

The commands for directly specifying rectangles are not supported.

OpenGL 1.3	**Safety Critical**
Rectf`(T x1, T y1, T x2, T y2)`	–
Rect{sid}`(T x1, T y1, T x2, T y2)`	–
Rect{sifd}v`(T v1[2], T v2[2])`	–

■ Although rectangle commands are used by existing Safety Critical applications, they can be readily emulated by rendering triangle strips. ❑

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 `TEXTURE` matrix; and no support for texture coordinate generation. The double-precision only commands **DepthRange**, **Frustum**, and **Ortho** are replaced with single-precision variants from the `OES_single_precision` extension.

OpenGL 1.3	**Safety Critical**
DepthRange `(clampd n, clampd f)`	◇
Viewport `(int x, int y, sizei w, sizei h)`	✓
MatrixMode `(enum mode)`	
`mode = MODELVIEW,PROJECTION`	✓
`mode = COLOR,TEXTURE`	–
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)`	
`PROJECTION` (2 deep)	✓
`MODELVIEW` (16 deep)	✓
`TEXTURE`	–
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 with the exception of texture matrices are kept since they are used by a large number of applications. Inclusion of the texture matrix stack will be considered for 1.1. 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. Texture transformation is not a requirement, and so it is not included. While the texgen command is useful, it is considered too much of an implementation burden (applications can implement it to some extent themselves). 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	Safety Critical
ClipPlane`(enum plane, const double *equation)`	–
GetClipPlane`(enum plane, double *equation)`	–
Enable/Disable`(CLIP_PLANE{0-5})`	–

■ User-specified clipping planes aren't required by current Safety Critical applications. User-specified clipping planes will be a candidate for 1.1. ❑

2.12 Current Raster Position

The concept of the current raster position for positioning pixel rectangles and bitmaps is supported. Current raster state and commands for setting the raster position are supported.

OpenGL 1.3	Safety Critical
RasterPos2{sifd}[v]`(T coords)`	–
RasterPos3f`(T coords)`	✓
RasterPos3fv`(T coords)`	–
RasterPos3{sid}[v]`(T coords)`	–
RasterPos4{sifd}[v]`(T coords)`	–

■ Bitmaps and pixel image primitives are supported. These primitives are positioned using the raster position command. Similar, to the **Color**, **Normal**, etc., commands, the most general floating-point form is supported. ❑

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`.

Only directional lights are supported. An implementation must support a minimum of 2 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 not supported. 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	Safety Critical
FrontFace`(enum mode)`	✓
Enable/Disable`(LIGHTING)`	✓
Enable/Disable`(LIGHT{0-1})`	✓
Materialf[v]`(enum face, enum pname, T param)`	
`face = FRONT_AND_BACK`	✓
`face = FRONT,BACK`	–
`pname = AMBIENT,DIFFUSE,SPECULAR,EMISSION,SHININESS`	✓
`pname = AMBIENT_AND_DIFFUSE`	✓
`pname = COLOR_INDEXES`	–
Materiali[v]`(enum face, enum pname, T param)`	–
GetMaterialfv`(enum face, enum pname, T *params)`	
`pname = AMBIENT,DIFFUSE,SPECULAR,EMISSION,SHININESS`	✓
`pname = COLOR_INDEXES`	–
GetMateriali[v]`(enum face, enum pname, int *params)`	–
GetMaterialf`(enum face, enum pname, float *params)`	–
Lightfv`(enum light, enum pname, T param)`	
`pname = AMBIENT,DIFFUSE,SPECULAR`	✓
`pname = POSITION`	✓
`pname = SPOT_CUTOFF,SPOT_DIRECTION,SPOT_EXPONENT`	–
`pname = CONSTANT_ATTENUATION`	–
`pname = LINEAR_ATTENUATION`	–
`pname = QUADRATIC_ATTENUATION`	–
Lightf`(enum light, enum pname, T param)`	–
Lighti[v]`(enum light, enum pname, T param)`	–
GetLightfv`(enum light, enum pname, T *params)`	
`pname = AMBIENT,DIFFUSE,SPECULAR`	✓
`pname = POSITION`	✓
`pname = SPOT_CUTOFF,SPOT_DIRECTION,SPOT_EXPONENT`	–
`pname = CONSTANT_ATTENUATION`	–
`pname = LINEAR_ATTENUATION`	–
`pname = QUADRATIC_ATTENUATION`	–

GetLighti[v]`(enum light, enum pname, int *params)`	–
GetLightf`(enum light, enum pname, float *params)`	–
LightModelf[v]`(enum pname, T param)`	
`pname = LIGHT_MODEL_AMBIENT`	✓
`pname = LIGHT_MODEL_TWO_SIDE`	–
`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, in safety critical applications, it is not used for complex cosmetic effects, or to model ”real-life” lighting. Rather, it might be used to add shading to a relief map, or to add a ”3D” feel to an instrument. To support this usage, only non-local lights are required (e.g. directional lights) in the Safety Critical profile. The minimum number of lights is reduced to 2, which greatly reduces the testing burden, while allowing for multiple lighting effects. Extensive research has not shown any need for more than 2 light sources. Support for secondary color is not required so it is not included. Local viewer and spot lights are not widely used - even in the workstation space, and is is removed to reduce testing burden. Two-sided lighting is not required by Safety Critical applications and removing this feature further simplifies certification tests. Scene ambient is retained since its default value is non-zero and there would be no method to disable it’s effect if it were not included.

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 not supported.

OpenGL 1.3	**Safety Critical**
Enable/Disable`(MULTISAMPLE)`	–

■ The Common profile specifies multisampling as optional. There is no demand for multisampling by current Safety Critical (2D) applications. Polygon smoothing or line smoothing along the edges is sufficient and more appropriate for current applications. Multisampling will be a candidate for 1.1. ❑

3.3 Points

Aliased and antialiased points are fully supported.

OpenGL 1.3	**Safety Critical**
PointSize`(float size)`	✓
Enable/Disable`(POINT_SMOOTH)`	✓

■ See below. ❑

3.4 Line Segments

Aliased and antialiased lines are fully supported. Line stippling is also fully supported.

OpenGL 1.3	Safety Critical
LineWidth `(float width)`	✓
Enable/Disable `(LINE_SMOOTH)`	✓
LineStipple `(int factor, ushort pattern)`	✓
Enable/Disable `(LINE_STIPPLE)`	✓

■ Antialiasing is important for visual quality, an important issue in many safety critical applications. 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 commonly used in safety critical applications. ❑

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 `POLYGON_SMOOTH`. Depth offset is supported in `FILL` mode only.

OpenGL 1.3	Safety Critical
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. Polygon modes are mostly used for debugging purposes, and are therefore not supported. Polygon smooth is as desirable as antialiasing for other primitives. However, its usefulness without support for polygon primitives and edge flag is very limited; Polygon smooth is therefore not supported. Face culling is important for eliminating unnecessary rasterization. Polygon stippling is commonly used in safety critical applications. Polygon offset for filled triangles is necessary for rendering coplanar and outline polygons and if not present requires either stencil buffers or application tricks. ❑

3.6 Pixel Rectangles

Support for drawing pixel rectangles is limited to the format `RGBA` and type `UNSIGNED_BYTE`. Limited **PixelStore** support is retained to allow different pack alignments for **ReadPixels** and unpack alignments for **Bitmap**, **DrawPixels** and **TexImage2D**. **PixelTransfer** modes and **PixelZoom** are not supported. The Imaging subset is not supported.

OpenGL 1.3	Safety Critical
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)	
format = RGBA type = UNSIGNED_BYTE	✓
all other combinations	–
PixelZoom(float xfactor, float yfactor)	–

■ The OpenGL 1.3 specification includes substantial support for operating on pixel images. Safety Critical applications require the ability to draw images directly in window coordinates, but with the constraint of minimizing the certification burden. To that end, the set of supported image formats and types is limited to one general format: RGBA with 8-bit components. There is minimal support for operating on images. The one exception is limited pixel storage mode support to allow different alignment options (other than the 4-byte OpenGL 1.0 default). **PixelZoom** and **PixelTransfer** are not used by Safety Critical applications.

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 fully supported.

OpenGL 1.3	Safety Critical
Bitmap`(sizei width, sizei height, float xorig, float yorig, float xmove, float ymove, const ubyte *bitmap)`	✓

■ The **Bitmap** command doesn't offer any functionality which is not covered by **DrawPixels**. It is supported because Safety Critical applications are using it to save memory usage on the host. ❑

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 formats are supported, listed in Table 3.1. Only the image type `UNSIGNED_BYTE` is supported. 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.

Additional restrictions apply to loading of previously loaded texture levels: *internalformat* must be identical to any previously loaded texture level. *width* must be $2n + 2(border)$, and if mipmapping is enabled must be a possible size for the level specified based on the size of width for other previously loaded levels. *height* must be $2n + 2(border)$, and if mipmapping is enabled must be a possible size for the level specified based on the size of height for other previously loaded levels. *border* must be identical to any previously loaded texture level. These restrictions hold true if any level of the active texture, including the one being loaded, have been previously loaded. These restrictions have no bearing if the level being loaded is the first texture loaded for the currently bound texture unit. If these restrictions are violated, an `INVALID_OPERATION` is generated.

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

CopyTexture, **CopyTexSubImage** and compressed textures are not supported.

OpenGL 1.3	Safety Critical
`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

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.

OpenGL 1.3	**Safety Critical**
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)	–
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)	–
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)	–
TexParameteri(enum target, enum pname, int 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{iv f[v]}(enum target, enum pname, T param)	–
GetTexParameterfv(enum target, enum pname, float *params)	–
GetTexParameteriv(enum target, enum pname, int *params)	
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	–
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	–
TexEnvfv(enum target, enum pname, float *params)	
pname = TEXTURE_ENV_COLOR	✓
pname = TEXTURE_ENV_MODE	–
pname = COMBINE_RGB,COMBINE_ALPHA	–
pname = SOURCE{012}_RGB,SOURCE{012}_ALPHA	–
pname = RGB_SCALE,ALPHA_SCALE	–
TexEnvi(enum target, enum pname, int 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	–
TexEnv{iv f}(enum target, enum pname, T param)	–
GetTexEnvfv(enum target, enum pname, float *params)	
pname = TEXTURE_ENV_COLOR	✓
pname = TEXTURE_ENV_MODE	–
pname = COMBINE_RGB,COMBINE_ALPHA	–
pname = SOURCE{012}_RGB,SOURCE{012}_ALPHA	–
pname = RGB_SCALE,ALPHA_SCALE	–
GetTexEnviv(enum target, enum pname, int *params)	
pname = TEXTURE_ENV_COLOR	–
pname = TEXTURE_ENV_MODE	✓
pname = COMBINE_RGB,COMBINE_ALPHA	–
pname = SOURCE{012}_RGB,SOURCE{012}_ALPHA	–
pname = RGB_SCALE,ALPHA_SCALE	–

■ Texturing with 2D images is a critical feature for mapping operations, and in some cases, for drawing text with arbitrary position and orientation. 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.

Texture deletion is not supported, as it is difficult to handle fragmentation and to meet real-time constraints at the same time. A real-time friendly texture deletion mechanism will be considered for 1.1.

To further facilitate the texture memory manager to meet real-time constraints, possible modifications of texture have been limited. Once a texture level is loaded, it cannot be modified in size. **TexImage2D** is restricted to allow only modifications of a loaded texture level which do not affect its internalformat, width, height or border.

A limited set of formats, types and internal formats is included. The RGB component ordering is always RGBA rather than BGRA since there is no real perceived advantage to using BGRA. The supported formats are the same as in the Common profile. The Common profiles support many low-resolution types to save memory. Safety Critical devices are not as memory constrained as mobile information devices. Textures with 8-bits per component are typical.

As opposed to the Common profile, copying from the framebuffer into textures is not supported, since this feature has not been demanded. If this functionality was added, ”render to texture” would be preferred for its efficiency.

Texture borders are not included, as the modern way is to use `CLAMP_TO_EDGE`. If proper transition/interpolation across tiled textures is necessary, a sub region of the texture $(2^n - 1)$ can be used. 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.

Internal Format	**External Format**	**Type**	**Bytes per Pixel**
`RGBA`	`RGBA`	`UNSIGNED_BYTE`	4
`RGB`	`RGB`	`UNSIGNED_BYTE`	3
`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

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. In the safety-critical space, texture compression is not an issue and is not included. In the safety-critical space, texture priority, and ”residence” are not relevant - in this space, it is very important to have repeatable, consistent execution time, and so most implementations will choose only to maintain ”resident” textures. ❑

3.9 Fog

Fog is not supported.

OpenGL 1.3	**Safety Critical**
Fog{if}[v] `(enum pname, T param)`	–
Enable/Disable `(FOG)`	–

■ The Common profile supports all fog modes. Current Safety Critical (2D) Applications are not using fog. Terrain rendering application are expected to use this feature in the future. Fog is a candidate for 1.1. ❑

Chapter 4

Per-Fragment Operations and the Framebuffer

4.1 Per-Fragment Operations

Most per-fragment operation functionalities are reduced.

Color index related operations and the imaging subset additions (**BlendColor** and **BlendEquation**) are not supported. Alpha test is supported for `LEQUAL` and `ALWAYS`. Blend func is supported for the following combinations:

Source Factor	**Destination Factor**
SRC_ALPHA	ONE_MINUS_SRC_ALPHA
SRC_ALPHA_SATURATE	ONE
ONE	ZERO

Table 4.1: Blend Factor Combinations

Depth test is supported for `LESS`, `LEQUAL`, and `ALWAYS`. Depth masking is supported. All other masking operations are not supported. Dithering and logic op is not supported. Scissor and stencil operations are fully supported. An implementation is not required to include a depth or stencil buffer.

OpenGL 1.3	**Safety Critical**
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)	

func = LEQUAL,ALWAYS	✓
func = *all other values*	–
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)	
func = LESS,LEQUAL,ALWAYS	✓
func = *all other values*	–
DepthMask(boolean flag)	✓
Enable/Disable(BLEND)	✓
BlendFunc(enum sfactor, enum dfactor)	
sfactor = SRC_ALPHA, dfactor = ONE_MINUS_SRC_ALPHA	✓
sfactor = SRC_ALPHA_SATURATE, dfactor = ONE	✓
sfactor = ONE, dfactor = ZERO	✓
all other combinations	–
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 and smoothing, other combinations of blend factors are removed to simplify certification tests. Additive blending will be a candidate for 1.1. Dithering is not required since frame buffers usually offer a resolution of 8 bits per component or more. By removing dithering, no implementation algorithm needs to be defined which would have to be tested for. Logic op is not used in the safety critical space. Its omission will simplify certification tests. The Common profile includes all masked operations. In the Safety Critical profile, masked operations other than **DepthMask** are not supported to simplify certification. **DepthMask** allows depth test without updating the depth buffer, which is useful for rendering transparent objects. In many cases, such as depth test, alpha test, blending, the number of potential modes has been reduced in order to limit the certification burden. Many modes are included in OpenGL 1.3 for completeness, and are not useful to most applications. Depth test is not only useful for correct depth occlusion, but also for visibility control of objects according to assigned priorities. The depth comparison function `LESS` is default and is the most commonly used function. The depth comparison function `LEQUAL` is useful to render screen aligned anti-aliased lines and points correctly without requiring polygon offset. The depth comparison function `ALWAYS` is useful for depth buffer shaping. In Safety Critical applications, it can be used to cut out complex instruments, and for filtering priority classes of objects. **DepthFunc**(`ALWAYS`) and **DepthMask** complement each other. ❑

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	**Safety Critical**
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 (see **DrawPixels**). The only format/type combination which is supported is format `RGBA` and type `UNSIGNED_BYTE`. **CopyPixels** is supported for color buffer only. **ReadBuffer** is not supported. Read operations return data from the default color buffer.

OpenGL 1.3	**Safety Critical**
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)`	
`type = COLOR`	✓
`type = DEPTH,STENCIL`	–

■ Reading the color buffer is useful for some applications and also provides a platform independent method for testing. Pixel copy performance is important some applications, and in addition, on those platforms that support offscreen PBuffers, it provides the fastest possible mechanism for copying pre-rendered portions of the screen into the current buffer. 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. **CopyPixels** is supported to enable fast copy from a pbuffer. This allows the hardware to be run at reduced clock speed to economize power consumption. **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	Safety Critical
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	Safety Critical
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	Safety Critical
FeedbackBuffer `(sizei size, enum type, float *buffer)`	–
PassThrough `(float token)`	–

■ Feedback is seldom used. ❑

5.4 Display Lists

Display lists are supported with the exception of **CallList**, **IsList** and **DeleteLists**. An implementation does not need to support recursive display list execution (`MAX_LIST_NESTING` is at least 1).

OpenGL 1.3	Safety Critical
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)`	–

■ Safety Critical applications using the Begin/End paradigm also use display lists to improve performance. On a system with multiple applications running simultaneously, drivers are implemented in such a way that one application cannot damage another. The drivers are rendering in indirect mode, which has a penalty on bandwidth. Using display lists allows to reduce bandwidth requirements to gain back performance.

CallList is not supported as the same functionality is already provided by **CallLists**. **IsList** is not supported as it is not useful for state preservation.

Desktop OpenGL includes **DeleteLists** to allow complex applications to reclaim display list memory at run-time. However, implementing **DeleteLists** requires ”real” memory management for handling the display list names, adding a burden to the driver implementation. Also, it is difficult to implement compacting or garbage-collection while respecting real-time constraints. Current Safety Critical applications rarely call **DeleteLists**, it is therefore feasible to omit **DeleteLists** and to push the memory management issues from the driver to the application. Inclusion of **DeleteLists** or a replacement with similar functionality will be considered for 1.1.

Nesting of display lists results in recursive execution of display lists. This is not desirable in Safety Critical applications, as it makes the execution time of a display list less deterministic, failing to meet a real-time constraint. The minimal maximum nesting depth is reduced to 1 to allow implementations without recursive display list execution. ❑

5.5 Flush and Finish

Flush and **Finish** are supported.

OpenGL 1.3	Safety Critical
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 related to supported features are retained. The implementation must specify precisely how the functionality referred to by each hint is implemented, including the semantics of `FASTEST`, `NICEST`, and in particular `DONT_CARE`.

OpenGL 1.3	Safety Critical
Hint `(enum target, enum mode)`	
`target = PERSPECTIVE_CORRECTION_HINT`	✓
`target = POINT_SMOOTH_HINT`	✓
`target = LINE_SMOOTH_HINT`	✓
`target = POLYGON_SMOOTH_HINT`	–
`target = FOG_HINT`	–
`target = TEXTURE_COMPRESSION_HINT`	–

■ Hints are supported because Safety Critical applications are typically developed on desktop systems where OpenGL implementations usually set the hint mode `DONT_CARE` (the default) equal to `FASTEST`. Safety Critical applications prefer to use `NICEST`, therefore the hints are being used. For certification purposes, an implementation must document the semantics of the hint modes. ❑

Chapter 6

State and State Requests

6.1 Querying GL State

State queries are supported for *static* and *dynamic* states, errors (see GL Errors) and strings. Query of derived dynamic states is not 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 the two-character profile identifier `SC` for the Safety Critical 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-SC 1.0`”
`EXTENSIONS`	as defined by OpenGL 1.3

Table 6.1: String State

Client and server attribute stacks are not supported by the profile; consequently, the commands **PushAttrib**, **PopAttrib**, **PushClientAttrib**, and **PopClientAttrib** are not supported.

OpenGL 1.3	**Safety Critical**
GetBooleanv `(enum pname, boolean *params)`	✓‡
GetClipPlane `(enum plane, double eqn[4])`	–
GetCompressedTexImage `(enum target, int level, void *image)`	–
GetDoublev `(enum pname, double *params)`	–
GetFloatv `(enum pname, float *params)`	✓‡
GetIntegerv `(enum pname, int *params)`	✓‡
GetLightfv `(enum light, enum pname, float *params)`	✓‡

GetMapifdv `(enum target, enum pname, T *params)`	–
GetMaterialfv `(enum face, enum pname, float *params)`	✓[‡]
GetMaterialiv `(enum face, enum pname, int *params)`	–
GetPixelMapf ui usv `(enum map, T *params)`	–
GetPointerv `(enum pname, void **params)`	✓[‡]
GetPolygonStipple `(ubyte *mask)`	✓
GetString `(enum name)`	✓
GetTexEnvfv `(enum env, enum pname, float *params)`	✓[‡]
GetTexEnviv `(enum env, enum pname, int *params)`	✓[‡]
GetTexGen{if}v `(enum coord, enum pname, T *params)`	–
GetTexImage `(enum target, int level, enum format, enum type, void *pixels)`	–
GetTexLevelParameter{if}v `(enum target, int level, enum pname, T *params)`	–
GetTexParameterfv `(enum target, enum pname, float *params)`	–
GetTexParameteriv `(enum target, enum pname, int *params)`	✓[‡]
IsEnabled `(enum cap)`	✓[‡]
IsTexture `(uint texture)`	–
IsList `(uint list)`	–
PushAttrib `(bitfield mask)`	–
PopAttrib `(void)`	–
PushClientAttrib `(bitfield mask)`	–
PopClientAttrib `(void)`	–

■ For small and monolithic Safety Critical applications, there should never be the need for the to query the OpenGL state. Anything which the application does that is based on a query of a static piece of data, for example, could (and should) easily have been built into the application at compile time, thus reducing the certification effort.

Large-Scale Safety Critical applications are composed of multiple modules, potentially from different vendors. The modules are certified individually, simplifying the certification effort of the integrated application.

Without a mechanism for state preservation, the vendors have to rely on conventions and completeness of API documentation with respect to state changes. To allow defensive implementation of middleware, either Gets or attribute stacks are required.

Gets are real-time friendlier: For hardware implementations, the queryable states can be shadowed. The dynamic states can be queried within a predicted time window. Gets do not require dynamic memory allocation. State memory management is pushed back to the application. Gets can also be helpful for certification tests.

Attribute stacks are easier to use and may be more efficient when many states have to be changed. Attribute stacks may be accelerated by hardware implementations. Attribute stacks have a stack size limit.

As the real-time requirements which are easily satisfiable by Gets outweigh the convenience and possible performance improvements of attribute stacks, the choice is to support Gets.

The Common profile as of version 1.1 includes dynamic gets as well. A difference to the Common profile is the query of matrix stack values. Code generators often use this feature to avoid keeping track of the current matrices. The Common profile will support query of matrix stacks through an extension.

Static gets ease conformance test porting and development of middleware for large-scale Safety Critical applications. The certification burden of static gets is minimal. Although the values returned by static gets could be obtained otherwise at build time, it is better to have a standard way to provide this information.

The string queries are retained as they provide important versioning, and extension information. ❏

6.2 State Tables

The tables also indicate which state variables are obtained with what commands. State variables that can be obtained using any of **GetBooleanv**, **GetIntegerv**, or **GetFloatv** are listed with just one of these commands - the one that is most appropriate given the type of data to be returned and the profile used. These state variables cannot be obtained using **IsEnabled**. However, state variables for which **IsEnabled** is listed as the query command can also be obtained using **GetBooleanv**, **GetIntegerv**, and **GetFloatv**. State variables for which any other command other than **IsEnabled** is listed as the query command can be obtained only by using that command. State appearing in *italics* indicates unnamed state. All state has initial values identical to those specified in OpenGL 1.3.

State	Exposed	Queriable	Command
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**	**Command**
`CURRENT_COLOR`	✓	✓	**GetFloatv**
`CURRENT_INDEX`	–	–	–
`CURRENT_TEXTURE_COORDS`	✓	✓	**GetFloatv**
`CURRENT_NORMAL`	✓	✓	**GetFloatv**
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`	✓	✓	**GetFloatv**
`CURERNT_RASTER_INDEX`	–	–	–
`CURRENT_RASTER_TEXTURE_COORDS`	✓	✓	**GetFloatv**
`CURRENT_RASTER_POSITION_VALID`	✓	–	–
`EDGE_FLAG`	–	–	–

Table 6.5: Current Values and Associated Data

State	**Exposed**	**Queriable**	**Command**
`CLIENT_ACTIVE_TEXTURE`	✓	✓	**GetIntegerv**
`VERTEX_ARRAY`	✓	✓	**IsEnabled**
`VERTEX_ARRAY_SIZE`	✓	✓	**GetIntegerv**
`VERTEX_ARRAY_STRIDE`	✓	✓	**GetIntegerv**
`VERTEX_ARRAY_TYPE`	✓	✓	**GetIntegerv**
`VERTEX_ARRAY_POINTER`	✓	✓	**GetPointerv**
`NORMAL_ARRAY`	✓	✓	**IsEnabled**
`NORMAL_ARRAY_STRIDE`	✓	✓	**GetIntegerv**
`NORMAL_ARRAY_TYPE`	✓	✓	**GetIntegerv**
`NORMAL_ARRAY_POINTER`	✓	✓	**GetPointerv**
`COLOR_ARRAY`	✓	✓	**IsEnabled**
`COLOR_ARRAY_SIZE`	✓	✓	**GetIntegerv**
`COLOR_ARRAY_STRIDE`	✓	✓	**GetIntegerv**
`COLOR_ARRAY_TYPE`	✓	✓	**GetIntegerv**
`COLOR_ARRAY_POINTER`	✓	✓	**GetPointerv**
`INDEX_ARRAY`	–	–	–
`INDEX_ARRAY_STRIDE`	–	–	–
`INDEX_ARRAY_TYPE`	–	–	–
`INDEX_ARRAY_POINTER`	–	–	–

Table 6.6: Vertex Array Data

State	**Exposed**	**Queriable**	**Command**
`TEXTURE_COORD_ARRAY`	✓	✓	**IsEnabled**
`TEXTURE_COORD_ARRAY_SIZE`	✓	✓	**GetIntegerv**
`TEXTURE_COORD_ARRAY_STRIDE`	✓	✓	**GetIntegerv**
`TEXTURE_COORD_ARRAY_TYPE`	✓	✓	**GetIntegerv**
`TEXTURE_COORD_ARRAY_POINTER`	✓	✓	**GetPointerv**
`EDGE_FLAG_ARRAY`	–	–	–
`EDGE_FLAG_ARRAY_STRIDE`	–	–	–
`EDGE_FLAG_ARRAY_POINTER`	–	–	–

Table 6.7: Vertex Array Data (cont.)

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

Table 6.8: Transformation State

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

Table 6.9: Coloring

State	**Exposed**	**Queriable**	**Command**
`LIGHTING`	✓	✓	**IsEnabled**
`COLOR_MATERIAL`	✓	✓	**IsEnabled**
`COLOR_MATERIAL_PARAMETER`	–	–	–
`COLOR_MATERIAL_FACE`	–	–	–
`AMBIENT (material)`	✓	✓	**GetMaterialfv**
`DIFFUSE (material)`	✓	✓	**GetMaterialfv**
`SPECULAR (material)`	✓	✓	**GetMaterialfv**
`EMISSION (material)`	✓	✓	**GetMaterialfv**
`SHININESS (material)`	✓	✓	**GetMaterialfv**
`LIGHT_MODEL_AMBIENT`	✓	✓	**GetFloatv**
`LIGHT_MODEL_LOCAL_VIEWER`	–	–	–
`LIGHT_MODEL_TWO_SIDE`	–	–	–
`LIGHT_MODEL_COLOR_CONTROL`	–	–	–
AMBIENT (light$_i$)	✓	✓	**GetLightfv**
DIFFUSE (light$_i$)	✓	✓	**GetLightfv**
SPECULAR (light$_i$)	✓	✓	**GetLightfv**
POSITION (light$_i$)	✓	✓	**GetLightfv**
`CONSTANT_ATTENUATION`	–	–	–
`LINEAR_ATTENUATION`	–	–	–
`QUADRATIC_ATTENUATION`	–	–	–
`SPOT_DIRECTION`	–	–	–
`SPOT_EXPONENT`	–	–	–
`SPOT_CUTOFF`	–	–	–
`LIGHT{0-1}`	✓	✓	**IsEnabled**
`COLOR_INDEXES`	–	–	–

Table 6.10: Lighting

State	**Exposed**	**Queriable**	**Command**
`POINT_SIZE`	✓	✓	**GetFloatv**
`POINT_SMOOTH`	✓	✓	**IsEnabled**
`LINE_WIDTH`	✓	✓	**GetFloatv**
`LINE_SMOOTH`	✓	✓	**IsEnabled**
`LINE_STIPPLE_PATTERN`	✓	✓	**GetIntegerv**
`LINE_STIPPLE_REPEAT`	✓	✓	**GetIntegerv**
`LINE_STIPPLE`	✓	✓	**IsEnabled**
`CULL_FACE`	✓	✓	**IsEnabled**
`CULL_FACE_MODE`	✓	✓	**GetIntegerv**
`FRONT_FACE`	✓	✓	**GetIntegerv**
`POLYGON_SMOOTH`	–	–	–
`POLYGON_MODE`	–	–	–
`POLYGON_OFFSET_FACTOR`	✓	✓	**GetFloatv**
`POLYGON_OFFSET_UNITS`	✓	✓	**GetFloatv**
`POLYGON_OFFSET_POINT`	–	–	–
`POLYGON_OFFSET_LINE`	–	–	–
`POLYGON_OFFSET_FILL`	✓	✓	**IsEnabled**
polygon stipple pattern	✓	✓	**GetPolygonStipple**
`POLYGON_STIPPLE`	✓	✓	**IsEnabled**

Table 6.11: Rasterization

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

Table 6.12: Multisampling

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

Table 6.13: Texture Objects

State	**Exposed**	**Queriable**	**Command**
`ACTIVE_TEXTURE`	✓	✓	**GetIntegerv**
`TEXTURE_ENV_MODE`	✓	✓	**GetTexEnviv**
`TEXTURE_ENV_COLOR`	✓	✓	**GetTexEnvfv**
`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_SCALE`	–	–	–

Table 6.14: Texture Environment and Generation

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

Table 6.15: Pixel Operations

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

Table 6.16: Framebuffer Control

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

State	Exposed	Queriable	Command
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	–	–	–
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 6.18: Pixels (cont.)

State	Exposed	Queriable	Command
`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 6.19: Pixels (cont.)

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

Table 6.20: Evaluators

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

Table 6.21: Hints

State	Exposed	Queriable	Command
`MAX_LIGHTS`	✓	✓	**GetIntegerv**
`MAX_CLIP_PLANES`	–	–	–
`MAX_COLOR_MATRIX_STACK_DEPTH`	–	–	–
`MAX_MODELVIEW_STACK_DEPTH`	✓	✓	**GetIntegerv**
`MAX_PROJECTION_STACK_DEPTH`	✓	✓	**GetIntegerv**
`MAX_TEXTURE_STACK_DEPTH`	–	–	–
`SUBPIXEL_BITS`	✓	✓	**GetIntegerv**
`MAX_3D_TEXTURE_SIZE`	–	–	–
`MAX_TEXTURE_SIZE`	✓	✓	**GetIntegerv**
`MAX_CUBE_MAP_TEXTURE_SIZE`	–	–	–
`MAX_PIXEL_MAP_TABLE`	–	–	–
`MAX_NAME_STACK_DEPTH`	–	–	–
`MAX_LIST_NESTING`	✓	✓	**GetIntegerv**
`MAX_EVAL_ORDER`	–	–	–
`MAX_VIEWPORT_DIMS`	✓	✓	**GetIntegerv**
`MAX_ATTRIB_STACK_DEPTH`	–	–	–
`MAX_CLIENT_ATTRIB_STACK_DEPTH`	–	–	–
Maximum size of a color table	–	–	–
Maximum size of the histogram table	–	–	–
`AUX_BUFFERS`	–	–	–
`RGBA_MODE`	–	–	–
`INDEX_MODE`	–	–	–
`DOUBLEBUFFER`	–	–	–
`ALIASED_POINT_SIZE_RANGE`	✓	✓	**GetFloatv**
`SMOOTH_POINT_SIZE_RANGE`	✓	✓	**GetFloatv**
`SMOOTH_POINT_SIZE_GRANULARITY`	✓	✓	**GetFloatv**
`ALIASED_LINE_WIDTH_RANGE`	✓	✓	**GetFloatv**
`SMOOTH_LINE_WIDTH_RANGE`	✓	✓	**GetFloatv**
`SMOOTH_LINE_WIDTH_GRANULARITY`	✓	✓	**GetFloatv**

Table 6.22: Implementation Dependent Values

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

Table 6.23: Implementation Dependent Values (cont.)

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

Table 6.24: Implementation Dependent Pixel Depths

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

Table 6.25: Miscellaneous

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 implementer. 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. ❑

Extension Name	**Extension Type**
`OES_single_precision`	core addition
`EXT_paletted_texture`	required extension
`EXT_shared_texture_palette`	optional extension

Table 7.1: Safety Critical Extension Disposition

7.1 Single-precision Commands

The `OES_single_precision` 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 Safety Critical 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)`

7.2 Paletted Textures

The `EXT_paletted_texture` extension is supported only for texture internal format `COLOR_INDEX8_EXT`. Only the subset matching the profile feature set is included in the Safety Critical profile.

EXT_paletted_texture	**Safety Critical**
TexImage2D`(enum target, int level, int internalFormat, sizei width, sizei height, int border, enum format, enum type, const void *pixels)`	
`internalFormat = COLOR_INDEX8_EXT, format = COLOR_INDEX, type = UNSIGNED_BYTE`	✓
GetTexLevelParameter{**if**}**v**`(enum target, int level, enum pname, T *params)`	–
ColorTableEXT`(enum target, enum internalformat, sizei width, enum format, enum type, const void *data)`	
`target = TEXTURE_2D`	✓
`target =` *all other values*	–
ColorSubTableEXT`(enum target, sizei start, sizei count, enum format, enum type, const void *data)`	
`target = TEXTURE_2D`	✓
`target =` *all other values*	–
GetColorTableEXT`(enum target, enum format, enum type, void *data)`	
`target = TEXTURE_2D`	✓
`target =` *all other values*	–
GetColorTableParameterivEXT`(enum target, enum pname, int *params)`	
`target = TEXTURE_2D`	✓
`target =` *all other values*	–
GetColorTableParameterfvEXT`(enum target, enum pname, float *params)`	–

■ The `EXT_paletted_texture` extension separates textures into indices and palette. It allows saving of texture memory and efficient modification of the palette without the need to update an entire texture. ❑

7.3 Shared Texture Palette

The support of `EXT_shared_texture_palette` extension is optional. If it is supported, only the subset matching the profile feature set is included in the Safety Critical profile.

EXT_shared_texture_palette	**Safety Critical**
Enable/Disable `(SHARED_TEXTURE_PALETTE_EXT)`	✓
IsEnabled `(SHARED_TEXTURE_PALETTE_EXT)`	✓
GetBooleanv `(enum pname, boolean *params)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓
GetFloatv `(enum pname, float *params)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓
GetIntegerv `(enum pname, int *params)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓
ColorTableEXT `(enum target, enum internalformat, sizei width, enum format, enum type, const void *data)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓
ColorSubTableEXT `(enum target, sizei start, sizei count, enum format, enum type, const void *data)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓
GetColorTableEXT `(enum target, enum format, enum type, void *data)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓
GetColorTableParameterivEXT `(enum target, enum pname, int *params)`	
`target = SHARED_TEXTURE_PALETTE_EXT`	✓

■ The `EXT_shared_texture_palette` extension is useful to update palettes shared between multiple textures. This feature is optional, as it is only used by specific (mapping) applications. ❑

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`. An additional enumerant `OES_SC_VERSION_x_y`, where `x` and `y` are the major and minor version numbers as described in Section 6.1, is 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_SC.z` where `.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 Safety Critical profile is 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:

Bill Marshall, Alt Software
Bruce Stockwell, Seaweed Systems
Chris Hall, Seaweed Systems
Claude Knaus, Esmertec
John Boal, Alt Software
John Jarvis, Alt Software
Mark Snyder, Quantum3D
Michal Krupa, Barco
Neal Countryman, Seaweed Systems
Neil Trevett, NVIDIA
Pete Daniel, Quantum3D
Steve Viggers, Alt Software
Ville Miettinen, Hybrid Graphics

The acknowledgements extend to the Common profiles working group for creating their specifications on which this document is based.

Appendix B

OES Extension Specifications

B.1 OES_single_precision

Name

OES_single_precision

Name Strings

GL_OES_single_precision

Contact

David Blythe (blythe ’at’ bluevoid.com)

Status

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

Version

Last Modifed Date: 28 June 2004
Author Revision : 0.5

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)

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

B.2 EXT_paletted_texture

Name

EXT_paletted_texture

Name Strings

GL_EXT_paletted_texture

Contact

Mark J. Kilgard, NVIDIA Corporation (mjk ’at’ nvidia.com)

Version

Last Modified Date: March 24, 2004
Revision: 1.4

Number

78

Support

Intel 810/815.

Mesa.

Microsoft software OpenGL implementation.

Selected NVIDIA GPUs: NV1x (GeForce 256, GeForce2, GeForce4 MX, GeForce4 Go, Quadro, Quadro2), NV2x (GeForce3, GeForce4 Ti, Quadro DCC, Quadro4 XGL), and NV3x (GeForce FX 5xxxx, Quadro FX 1000/2000/3000). NV3 (Riva 128) and NV4 (TNT, TNT2) GPUs and NV4x GPUs do NOT support this functionality (no hardware support). Future NVIDIA GPU designs will no longer support paletted textures.

S3 ProSavage, Savage 2000.

3Dfx Voodoo3, Voodoo5.

3Dlabs GLINT.

Dependencies

GL_EXT_paletted_texture shares routines and enumerants with GL_SGI_color_table with the minor modification that EXT replaces SGI. In all other ways these calls should function in the same manner and the enumerant values should be identical. The portions of GL_SGI_color_table that are used are:

ColorTableSGI, GetColorTableSGI, GetColorTableParameterivSGI,

```
                GetColorTableParameterfvSGI.
                COLOR_TABLE_FORMAT_SGI, COLOR_TABLE_WIDTH_SGI,
                COLOR_TABLE_RED_SIZE_SGI, COLOR_TABLE_GREEN_SIZE_SGI,
                COLOR_TABLE_BLUE_SIZE_SGI, COLOR_TABLE_ALPHA_SIZE_SGI,
                COLOR_TABLE_LUMINANCE_SIZE_SGI, COLOR_TABLE_INTENSITY_SIZE_SGI.

    Portions of GL_SGI_color_table which are not used in
    GL_EXT_paletted_texture are:
                CopyColorTableSGI, ColorTableParameterivSGI,
                ColorTableParameterfvSGI.
                COLOR_TABLE_SGI, POST_CONVOLUTION_COLOR_TABLE_SGI,
                POST_COLOR_MATRIX_COLOR_TABLE_SGI, PROXY_COLOR_TABLE_SGI,
                PROXY_POST_CONVOLUTION_COLOR_TABLE_SGI,
                PROXY_POST_COLOR_MATRIX_COLOR_TABLE_SGI, COLOR_TABLE_SCALE_SGI,
                COLOR_TABLE_BIAS_SGI.

    EXT_paletted_texture can be used in conjunction with EXT_texture3D.
    EXT_paletted_texture modifies TexImage3DEXT to accept paletted image
    data and allows TEXTURE_3D_EXT and PROXY_TEXTURE_3D_EXT to be used a
    targets in the color table routines.  If EXT_texture3D is unsupported
    then references to 3D texture support in this spec are invalid and
    should be ignored.

    EXT_paletted_texture can be used in conjunction with
    ARB_texture_cube_map.  EXT_paletted_texture modifies TexImage2D
    to accept paletted image data and allows TEXTURE_CUBE_MAP_ARB, and
    PROXY_TEXTURE_CUBE_MAP_ARB to be used a targets in the color table
    routines.  If ARB_texture_cube_map is unsupported then references
    to cube map texture support in this spec are invalid and should be
    ignored.

Overview

    EXT_paletted_texture defines new texture formats and new calls to
    support the use of paletted textures in OpenGL.  A paletted texture is
    defined by giving both a palette of colors and a set of image data which
    is composed of indices into the palette.  The paletted texture cannot
    function properly without both pieces of information so it increases the
    work required to define a texture.  This is offset by the fact that the
    overall amount of texture data can be reduced dramatically by factoring
    redundant information out of the logical view of the texture and placing
    it in the palette.

    Paletted textures provide several advantages over full-color textures:

    * As mentioned above, the amount of data required to define a
    texture can be greatly reduced over what would be needed for full-color
    specification.  For example, consider a source texture that has only 256
    distinct colors in a 256 by 256 pixel grid.  Full-color representation
    requires three bytes per pixel, taking 192K of texture data.  By putting
    the distinct colors in a palette only eight bits are required per pixel,
    reducing the 192K to 64K plus 768 bytes for the palette.  Now add an
```

alpha channel to the texture. The full-color representation increases by 64K while the paletted version would only increase by 256 bytes. This reduction in space required is particularly important for hardware accelerators where texture space is limited.

* Paletted textures allow easy reuse of texture data for images which require many similar but slightly different colored objects. Consider a driving simulation with heavy traffic on the road. Many of the cars will be similar but with different color schemes. If full-color textures are used a separate texture would be needed for each color scheme, while paletted textures allow the same basic index data to be reused for each car, with a different palette to change the final colors.

* Paletted textures also allow use of all the palette tricks developed for paletted displays. Simple animation can be done, along with strobing, glowing and other palette-cycling effects. All of these techniques can enhance the visual richness of a scene with very little data.

IP Status

None.

New Procedures and Functions

```
void ColorTableEXT(
    enum target,
    enum internalFormat,
    sizei width,
    enum format,
    enum type,
    const void *data);

void ColorSubTableEXT(
    enum target,
    sizei start,
    sizei count,
    enum format,
    enum type,
    const void *data);

void GetColorTableEXT(
    enum target,
    enum format,
    enum type,
    void *data);

void GetColorTableParameterivEXT(
    enum target,
    enum pname,
    int *params);
    void GetColorTableParameterfvEXT(
        enum target,
        enum pname,
        float *params);
```

New Tokens

Accepted by the internalformat parameter of TexImage1D, TexImage2D and TexImage3DEXT:

COLOR_INDEX1_EXT	0x80E2
COLOR_INDEX2_EXT	0x80E3
COLOR_INDEX4_EXT	0x80E4
COLOR_INDEX8_EXT	0x80E5
COLOR_INDEX12_EXT	0x80E6
COLOR_INDEX16_EXT	0x80E7

Accepted by the pname parameter of GetColorTableParameterivEXT and GetColorTableParameterfvEXT:

COLOR_TABLE_FORMAT_EXT	0x80D8
COLOR_TABLE_WIDTH_EXT	0x80D9
COLOR_TABLE_RED_SIZE_EXT	0x80DA
COLOR_TABLE_GREEN_SIZE_EXT	0x80DB
COLOR_TABLE_BLUE_SIZE_EXT	0x80DC
COLOR_TABLE_ALPHA_SIZE_EXT	0x80DD
COLOR_TABLE_LUMINANCE_SIZE_EXT	0x80DE
COLOR_TABLE_INTENSITY_SIZE_EXT	0x80DF

Accepted by the value parameter of GetTexLevelParameter{if}v:

TEXTURE_INDEX_SIZE_EXT	0x80ED

Accepted by the target parameter of ColorTableEXT, GetColorTableParameterivEXT, and GetColorTableParameterfvEXT:

TEXTURE_1D	0x0DE0
TEXTURE_2D	0x0DE1
TEXTURE_3D_EXT	0x806F
TEXTURE_CUBE_MAP_ARB	0x8513
PROXY_TEXTURE_1D	0x8063
PROXY_TEXTURE_2D	0x8064
PROXY_TEXTURE_3D_EXT	0x8070
PROXY_TEXTURE_CUBE_MAP_ARB	0x851B

Accepted by the target parameter of ColorSubTableEXT and GetColorTableEXT:

TEXTURE_1D	0x0DE0
TEXTURE_2D	0x0DE1
TEXTURE_3D_EXT	0x806F
TEXTURE_CUBE_MAP_ARB	0x8513

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

None

Additions to Chapter 3 of the GL Specification (Rasterization)

Section 3.6.4, 'Pixel Transfer Operations,' subsection 'Color Index Lookup,'

Point two is modified from 'The groups will be loaded as an image into texture memory' to 'The groups will be loaded as an image into texture memory and the internalformat parameter is not one of the color index formats from table 3.8.'

Section 3.8, 'Texturing,' subsection 'Texture Image Specification' is modified as follows:

The portion of the first paragraph discussing interpretation of format, type and data is split from the portion discussing target, width and height. The target, width and height section now ends with the sentence 'Arguments width and height specify the image's width and height.'

The format, type and data section is moved under a subheader 'Direct Color Texture Formats' and begins with 'If internalformat is not one of the color index formats from table 3.8,' and continues with the existing text through the internalformat discussion.

After that section, a new section 'Paletted Texture Formats' has the text:

If format is given as COLOR_INDEX then the image data is composed of integer values representing indices into a table of colors rather than colors themselves. If internalformat is given as one of the color index formats from table 3.8 then the texture will be stored internally as indices rather than undergoing index-to-RGBA mapping as would previously have occurred. In this case the only valid values for type are BYTE, UNSIGNED_BYTE, SHORT, UNSIGNED_SHORT, INT and UNSIGNED_INT.

The image data is unpacked from memory exactly as for a DrawPixels command with format of COLOR_INDEX for a context in color index mode. The data is then stored in an internal format derived from internalformat. In this case the only legal values of internalformat are COLOR_INDEX1_EXT, COLOR_INDEX2_EXT, COLOR_INDEX4_EXT, COLOR_INDEX8_EXT, COLOR_INDEX12_EXT and COLOR_INDEX16_EXT and the internal component resolution is picked according to the index resolution specified by internalformat. Any excess precision in the data is silently truncated to fit in the internal component precision.

An application can determine whether a particular implementation supports a particular paletted format (or any paletted formats at all) by attempting to use the paletted format with a proxy target. TEXTURE_INDEX_SIZE_EXT will be zero if the implementation cannot support the texture as given.

An application can determine an implementation's desired

> format for a particular paletted texture by making a TexImage call with COLOR_INDEX as the internalformat, in which case target must be a proxy target. After the call the application can query TEXTURE_INTERNAL_FORMAT to determine what internal format the implementation suggests for the texture image parameters. TEXTURE_INDEX_SIZE_EXT can be queried after such a call to determine the suggested index resolution numerically. The index resolution suggested by the implementation does not have to be as large as the input data precision. The resolution may also be zero if the implementation is unable to support any paletted format for the given texture image.

Table 3.8 should be augmented with a column titled 'Index bits.' All existing formats have zero index bits. The following formats are added with zeroes in all existing columns:

Name	Index bits
COLOR_INDEX1_EXT	1
COLOR_INDEX2_EXT	2
COLOR_INDEX4_EXT	4
COLOR_INDEX8_EXT	8
COLOR_INDEX12_EXT	12
COLOR_INDEX16_EXT	16

At the end of the discussion of level the following text should be added:

> All mipmapping levels share the same palette. If levels are created with different precision indices then their internal formats will not match and the texture will be inconsistent, as discussed above.

In the discussion of internalformat for CopyTexImage{12}D, at end of the sentence specifying that 1, 2, 3 and 4 are illegal there should also be a mention that paletted internalformat values are illegal.

At the end of the width, height, format, type and data section under TexSubImage there should be an additional sentence:

> If the target texture has an color index internal format then format may only be COLOR_INDEX.

At the end of the first paragraph describing TexSubImage and CopyTexSubImage the following sentence should be added:

> If the target of a CopyTexSubImage is a paletted texture image then INVALID_OPERATION is returned.

After the Alternate Image Specification Commands section, a new 'Palette Specification Commands' section should be added.

> Paletted textures require palette information to translate indices into full colors. The command

```
  void ColorTableEXT(enum target, enum internalformat, sizei width,
          enum format, enum type, const void *data);
```

is used to specify the format and size of the palette for paletted textures. target specifies which texture is to have its palette changed and may be one of TEXTURE_1D, TEXTURE_2D, PROXY_TEXTURE_1D, PROXY_TEXTURE_2D, TEXTURE_3D_EXT, PROXY_TEXTURE_3D_EXT, TEXTURE_CUBE_MAP_ARB, or PROXY_TEXTURE_CUBE_MAP_ARB. internalformat specifies the desired format and resolution of the palette when in its internal form. internalformat can be any of the non-index values legal for TexImage internalformat although implementations are not required to support palettes of all possible formats. width controls the size of the palette and must be a power of two greater than or equal to one. format and type specify the number of components and type of the data given by data. format can be any of the formats legal for DrawPixels although implementations are not required to support all possible formats. type can be any of the types legal for DrawPixels except GL_BITMAP.

Data is taken from memory and converted just as if each palette entry were a single pixel of a 1D texture. Pixel unpacking and transfer modes apply just as with texture data. After unpacking and conversion the data is translated into a internal format that matches the given format as closely as possible. An implementation does not, however, have a responsibility to support more than one precision for the base formats.

If the palette's width is greater than the range of the color indices in the texture data then some of the palettes entries will be unused. If the palette's width is less than the range of the color indices in the texture data then the most-significant bits of the texture data are ignored and only the appropriate number of bits of the index are used when accessing the palette.

Specifying a proxy target causes the proxy texture's palette to be resized and its parameters set but no data is transferred or accessed. If an implementation cannot handle the palette data given in the call then the color table width and component resolutions are set to zero.

Portions of the current palette can be replaced with

```
  void ColorSubTableEXT(enum target, sizei start, sizei count,
          enum format, enum type, const void *data);
```

target can be any of the non-proxy values legal for ColorTableEXT. start and count control which entries of the palette are changed out of the range allowed by the internal format used for the palette indices. count is silently clamped so that all modified entries all within the legal range. format and type can be any of the values legal for ColorTableEXT. The data is treated as a 1D texture just as in ColorTableEXT.

In the 'Texture State and Proxy State' section the sentence fragment beginning 'six integer values describing the resolutions...' should be changed to refer to seven integer values, with the seventh being the index resolution.

Palette data should be added in as a third category of texture state.

After the discussion of properties, the following should be added:

> Next there is the texture palette. All textures have a palette, even if their internal format is not color index. A texture's palette is initially one RGBA element with all four components set to 1.0.

The sentence mentioning that proxies do not have image data or properties should be extended with 'or palettes.'

The sentence beginning 'If the texture array is too large' describing the effects of proxy failure should change to read:

> If the implementation is unable to handle the texture image data the proxy width, height, border width and component resolutions are set to zero. This situation can occur when the texture array is too large or an unsupported paletted format was requested.

Additions to Chapter 4 of the GL Specification (Per-Fragment Operations and the Framebuffer)

None

Additions to Chapter 5 of the GL Specification (Special Functions)

Section 5.4, 'Display Lists' is modified as follows:

Include PROXY_TEXTURE_1D, PROXY_TEXTURE_2D, PROXY_TEXTURE_3D, and PROXY_TEXTURE_CUBE_MAP_ARB in the list of tokens for which ColorTableEXT is executed immediately.

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

In the section on GetTexImage, the sentence saying 'The components are assigned among R, G, B and A according to' should be changed to be

> If the internal format of the texture is not a color index format then the components are assigned among R, G, B, and A according to Table 6.1. Specifying COLOR_INDEX for format in this case will generate the error INVALID_ENUM. If the internal format of the texture is color index then the components are handled in one of two ways depending on the value of format. If format is not COLOR_INDEX, the texture's indices are passed through the texture's palette and the resulting components are assigned among R, G, B, and A according to Table 6.1. If format is COLOR_INDEX then the data is treated as single components and the palette indices are returned. Components are taken starting...

Following the GetTexImage section there should be a new section:

GetColorTableEXT is used to get the current texture palette.

```
  void GetColorTableEXT(enum target, enum format, enum type, void *data);
```

GetColorTableEXT retrieves the texture palette of the texture given by target. target can be any of the non-proxy targets valid for ColorTableEXT. format and type are interpreted just as for ColorTableEXT. All textures have a palette by default so GetColorTableEXT will always be able to return data even if the internal format of the texture is not a color index format.

Palette parameters can be retrieved using

```
  void GetColorTableParameterivEXT(enum target, enum pname, int *params);
  void GetColorTableParameterfvEXT(enum target, enum pname, float *params);
```

target specifies the texture being queried and pname controls which parameter value is returned. Data is returned in the memory pointed to by params.

Querying COLOR_TABLE_FORMAT_EXT returns the internal format requested by the most recent ColorTableEXT call or the default. COLOR_TABLE_WIDTH_EXT returns the width of the current palette. COLOR_TABLE_RED_SIZE_EXT, COLOR_TABLE_GREEN_SIZE_EXT, COLOR_TABLE_BLUE_SIZE_EXT and COLOR_TABLE_ALPHA_SIZE_EXT return the actual size of the components used to store the palette data internally, not the size requested when the palette was defined.

Table 6.11, "Texture Objects" should have a line appended for TEXTURE_INDEX_SIZE_EXT:

```
TEXTURE_INDEX_SIZE_EXT      n x Z+  GetTexLevelParameter 0  xD
                            texture image i’s index resolution   3.8     -
```

New State

In table 6.16, Texture Objects, p. 224, add the following:

Get Value	Type	Get Command	Initial Value
TEXTURE_1D	I	GetColorTableEXT	empty
TEXTURE_2D	I	GetColorTableEXT	empty
TEXTURE_3D	I	GetColorTableEXT	empty
TEXTURE_CUBE_MAP	I	GetColorTableEXT	empty
COLOR_TABLE_FORMAT_EXT	2x4xZn	GetColorTableParameterivEXT	RGBA
COLOR_TABLE_WIDTH_EXT	2x4xZ+	GetColorTableParameteriv	0
COLOR_TABLE_x_SIZE_EXT	6x2x4xZ+	GetColorTableParameteriv	0
TEXTURE_INDEX_SIZE_EXT	nxZ+	GetTexLevelParameter	0

Get Value	Description	Sec.	Attribute
TEXTURE_1D	1D palette	3.8	-
TEXTURE_2D	2D palette	3.8	-
TEXTURE_3D	3D palette	3.8	-
TEXTURE_CUBE_MAP	cube map palette	3.8	-
COLOR_TABLE_FORMAT_EXT	paletted texture formats	3.8	-
COLOR_TABLE_WIDTH_EXT	paletted texture width	3.8	-
COLOR_TABLE_x_SIZE_EXT	paletted texture component sizes	3.8	-
TEXTURE_INDEX_SIZE_EXT	texture image's index resolution	3.8	-

New Implementation Dependent State

None

Revision History

Original draft, revision 0.5, December 20, 1995 (drewb) Created

Minor revisions and clarifications, revision 0.6, January 2, 1996 (drewb)
Replaced all request-for-comment blocks with final text based on implementation.

Minor revisions and clarifications, revision 0.7, Feburary 5, 1996 (drewb)
Specified the state of the palette color information when existing data is replaced by new data.

Clarified behavior of TexPalette on inconsistent textures.

Major changes due to ARB review, revision 0.8, March 1, 1996 (drewb)
Switched from using TexPaletteEXT and GetTexPaletteEXT to using SGI's ColorTableEXT routines. Added ColorSubTableEXT so equivalent functionality is available.

Allowed proxies in all targets.

Changed PALETTE?_EXT values to COLOR_INDEX?_EXT. Added support for one and two bit palettes. Removed PALETTE_INDEX_EXT in favor of COLOR_INDEX.

Decoupled palette size from texture data type. Palette size is controlled only by ColorTableEXT.

Changes due to ARB review, revision 1.0, May 23, 1997 (drewb)
Mentioned texture3D.

Defined TEXTURE_INDEX_SIZE_EXT.

Allowed implementations to return an index size of zero to indicate no support for a particular format.

Allowed usage of GL_COLOR_INDEX as a generic format in proxy queries for determining an optimal index size for a particular texture.

Disallowed CopyTexImage and CopyTexSubImage to paletted formats.

Deleted mention of index transfer operations during GetTexImage with paletted formats.

Changes due to ARB_texture_cube_map, revision 1.1, June 27, 2002 (Mark Kilgard)
Add language to section 5.4 about proxy texture tokens for ColorTable executing immediately.

Document ARB_texture_cube_map interactions.

Document texture target usage for ColorTable API.

Add "New State" section with table and "New Implementation Dependent State" sections.

Changes when incorporating into the registry, September 4, 2002 (Jon Leech)
Added missing IP Status / Contact fields (without bumping the revision) and incorporated Mark’s changes into the registry.

Changes, revision 1.4, March 24, 2004 (Mark Kilgard)
Document vendor support for this extension; note that future NVIDIA GPU designs will not support this extension.

B.3 EXT_shared_texture_palette

Name

EXT_shared_texture_palette

Name Strings

GL_EXT_shared_texture_palette

Contact

Jon Leech, SGI (ljp ’at’ sgi.com)
Mark J. Kilgard, NVIDIA Corporation (mjk ’at’ nvidia.com)

Version

Last Modified Date: March 24, 2004
Revision: 1.4

Number

141

Support

Mesa.

Selected NVIDIA GPUs: NV1x (GeForce 256, GeForce2, GeForce4 MX, GeForce4 Go, Quadro, Quadro2), NV2x (GeForce3, GeForce4 Ti, Quadro DCC, Quadro4 XGL), and NV3x (GeForce FX 5xxxx, Quadro FX 1000/2000/3000). NV3 (Riva 128) and NV4 (TNT, TNT2) GPUs and NV4x GPUs do NOT support this functionality (no hardware support). Future NVIDIA GPU designs will no longer support paletted textures.

S3 ProSavage, Savage 2000.

3Dfx Voodoo3, Voodoo5.

3Dlabs GLINT.

Dependencies

EXT_paletted_texture is required.

Overview

EXT_shared_texture_palette defines a shared texture palette which may be used in place of the texture object palettes provided by EXT_paletted_texture. This is useful for rapidly changing a palette common to many textures, rather than having to reload the new palette for each texture. The extension acts as a switch, causing all lookups

```
    that would normally be done on the texture's palette to instead use the
    shared palette.

IP Status

    None.

Issues

    *  Do we want to use a new <target> to ColorTable to specify the
       shared palette, or can we just infer the new target from the
       corresponding Enable?

    *  A future extension of larger scope might define a "texture palette
       object" and bind these objects to texture objects dynamically, rather
       than making palettes part of the texture object state as the current
       EXT_paletted_texture spec does.

    *  Should there be separate shared palettes for 1D, 2D, and 3D
       textures?

       Probably not; palette lookups have nothing to do with the
       dimensionality of the texture. If multiple shared palettes
       are needed, we should define palette objects.

    *  There's no proxy mechanism for checking if a shared palette can
       be defined with the requested parameters. Will it suffice to
       assume that if a texture palette can be defined, so can a shared
       palette with the same parameters?

    *  The changes to the spec are based on changes already made for
       EXT_paletted_texture, which means that all three documents must
       be referred to. This is quite difficult to read.

    *  The changes to section 3.8.6, defining how shared palettes are
       enabled and disabled, might be better placed in section 3.8.1.
       However, the underlying EXT_paletted_texture does not appear to
       modify these sections to define exactly how palette lookups are
       done, and it's not clear where to put the changes.

    *  How does the shared texture palette interact with multitexture
       support?  There is a single global shared texture palette that
       all texture units utilize (as opposed to a shared texture palette
       per texture unit).

New Procedures and Functions

    None

New Tokens

    Accepted by the <pname> parameters of GetBooleanv, GetIntegerv,
```

GetFloatv, GetDoublev, IsEnabled, Enable, Disable, ColorTableEXT, ColorSubTableEXT, GetColorTableEXT, GetColorTableParameterivEXT, and GetColorTableParameterfd EXT:

```
SHARED_TEXTURE_PALETTE_EXT              0x81FB
```

Additions to Chapter 2 of the 1.1 Specification (OpenGL Operation)

None

Additions to Chapter 3 of the 1.1 Specification (Rasterization)

Section 3.8, ’Texturing,’ subsection ’Texture Image Specification’ is modified as follows:

In the Palette Specification Commands section, the sentence beginning ’target specifies which texture is to’ should be changed to:

> target specifies the texture palette or shared palette to be changed, and may be one of TEXTURE_1D, TEXTURE_2D, PROXY_TEXTURE_1D, PROXY_TEXTURE_2D, TEXTURE_3D_EXT, PROXY_TEXTURE_3D_EXT, or SHARED_TEXTURE_PALETTE_EXT.

In the ’Texture State and Proxy State’ section, the sentence beginning ’A texture’s palette is initially...’ should be changed to:

> There is also a shared palette not associated with any texture, which may override a texture palette. (Even when multiple texture units are available, there is still only a single shared texture palette.) All palettes are initially...

Section 3.8.6, ’Texture Application’ is modified by appending the following:

Use of the shared texture palette is enabled or disabled using the generic Enable or Disable commands, respectively, with the symbolic constant SHARED_TEXTURE_PALETTE_EXT.

The required state is one bit indicating whether the shared palette is enabled or disabled. In the initial state, the shared palettes is disabled.

Additions to Chapter 4 of the 1.1 Specification (Per-Fragment Operations and the Frame buffer)

Additions to Chapter 5 of the 1.1 Specification (Special Functions)

Additions to Chapter 6 of the 1.1 Specification (State and State Requests)

In the section on GetTexImage, the sentence beginning ’If format is not COLOR_INDEX...' should be changed to:

> If format is not COLOR_INDEX, the texture's indices are passed through the texture's palette, or the shared palette if one is enabled, and the resulting components are assigned among R, G, B, and A according to Table 6.1.

In the GetColorTable section, the first sentence of the second paragraph should be changed to read:

> GetColorTableEXT retrieves the texture palette or shared palette given by target.

The first sentence of the third paragraph should be changed to read:

> Palette parameters can be retrieved using
> ```
> void GetColorTableParameterivEXT(enum target, enum pname, int *params);
> void GetColorTableParameterfvEXT(enum target, enum pname, float *params);
> ```
> target specifies the texture palette or shared palette being queried and pname controls which parameter value is returned.

Additions to the GLX Specification

None

New State

Get Value	Type	Get Command	Initial Value
SHARED_TEXTURE_PALETTE_EXT	B	IsEnabled	False
SHARED_TEXTURE_PALETTE_EXT	I	GetColorTableEXT	empty
COLOR_TABLE_FORMAT_EXT	Zn	GetColorTableParameterivEXT	RGBA
COLOR_TABLE_WIDTH_EXT	Z+	GetColorTableParameteriv	0
COLOR_TABLE_x_SIZE_EXT	6xZ+	GetColorTableParameteriv	0

Get Value	Type	Description	Sec	Attribute
SHARED_TEXTURE_PALETTE_EXT	B	shared texture palette enable	3.8.6	texture/enable
SHARED_TEXTURE_PALETTE_EXT	I	shared texture palette table	3.8	-
COLOR_TABLE_FORMAT_EXT	Zn	shared texture palette format	3.8	-
COLOR_TABLE_WIDTH_EXT	Z+	shared texture palette width	3.8	-
COLOR_TABLE_x_SIZE_EXT	6xZ+	shared texture palette component sizes	3.8	-

New Implementation Dependent State

None

```
Revision History

    September 4, 2002 - Add missing IP Status / Contact fields
    (without bumping the revision) and incorporated Mark's changes
    into the registry. (Jon Leech)

    July 10, 2002 (version 1.3) - Added "New State" tables entries.
    Clarify that there is a single global shared texture palette,
    rather than a per-texture unit palette when multitexture is
    available. (Mark Kilgard)

    March 24, 2004 (version 1.4) - Document vendor support for this
    extension; note that future NVIDIA GPU designs will not support this
    extension. (Mark Kilgard)
```